Ondokuz Mayıs Üniversitesi
Eğitim Bilimleri Enstitüsü
Bilgisayar Ve Öğretim Teknolojileri Eğitimi
Anabilim Dalı

GÖRSEL PROGRAMLAMA ORTAMI İLE ÖĞRETİMİN ÖĞRENCİLERİN BİLGİSAYAR PROGRAMLAMAYI ÖĞRENMESİNE VE PROGRAMLAMAYA KARŞI TUTUMLARINA ETKİSİNİN İNCELENMESİ

Hazırlayan :
Mehmet Fatih YİĞİT

Danışman :
Prof. Dr. Mustafa BAŞER

Yüksek Lisans Tezi

Samsun, 2016

Ondokuz Mayıs Üniversitesi
Eğitim Bilimleri Enstitüsü
Bilgisayar Ve Öğretim Teknolojileri Eğitimi
Anabilim Dalı

GÖRSEL PROGRAMLAMA ORTAMI İLE ÖĞRETİMİN ÖĞRENCİLERİN BİLGİSAYAR PROGRAMLAMAYI ÖĞRENMESİNE VE PROGRAMLAMAYA KARŞI TUTUMLARINA ETKİSİNİN İNCELENMESİ

Hazırlayan :
Mehmet Fatih YİĞİT

Danışman :
Prof. Dr. Mustafa BAŞER

Yüksek Lisans Tezi

Samsun, 2016

KABUL VE ONAY

Mehmet Fatih YİĞİT tarafından hazırlanan "Görsel Programlama Öğretiminin Öğrencilerin Bilgisayar Programlamayı Öğrenmesine Ve Programlamaya Karşı Tutumlarına Etkisinin İncelenmesi" başlıklı bu çalışma 17/06/2016 tarihinde yapılan savunma sınavı sonucunda oybirliğiyle/oyçokluğuyla başarılı bulunarak jürimiz tarafından Yüksek Lisans Tezi olarak kabul edilmiştir.

Başkan : Prof. Dr. Mustafa Başer

Üye : Doç. Dr. Recep Çakır

Üye : Doç. Dr. Sönmez Kamuk

Yukarıdaki imzaların adı geçen öğretim üyelerine ait olduğunu onaylarım.

17/06/2016

BİLİMSEL ETİK BİLDİRİMİ

Hazırladığım Yüksek Lisans tezinin, proje aşamasından sonuçlanmasına kadarki süreçte bilimsel etiğe ve akademik kurallara özenle riayet ettiğimi, tez içindeki tüm bilgileri bilimsel ahlak ve gelenek çerçevesinde elde ettiğimi, tez yazım kurallarına uygun olarak hazırladığım bu çalışmamda doğrudan veya dolaylı olarak yaptığım her alıntıya kaynak gösterdiğimi ve yararlandığım eserlerin kaynakçada gösterilenlerden oluştuğunu taahhüt ederim.

17/06/2016

Mehmet Fatih YİĞİT

ÖZET

Öğrencinin Adı - Soyadı	Mehmet Fatih YİĞİT
Anabilim Dalı	Bilgisayar ve Öğretim Teknolojileri Eğitimi
Danışmanın Adı	Prof. Dr. Mustafa BAŞER
Tezin Adı	Görsel Programlama Ortamı ile Öğretimin Öğrencilerin Bilgisayar Programlamayı Öğrenmesine ve Programlamaya Karşı Tutumlarına Etkisinin İncelenmesi

Bu çalışma görsel porgramlama ortamı ile öğretimin öğrencilerin programlamayı öğrenmelerine ve programlamaya karşı tutumlarına etksini incelemek amacıyla yapılmıştır. Çalışma 2015-2016 Eğitim-Öğretim yılı güz döneminde Ondokuz Mayıs Üniversitesi Bilgisayar ve Öğretim Teknolojileri Eğitimi Bölümü'nde öğretim gören 42 ikinci sınıf öğrencisi üzerinde "Programlama Dilleri I" dersi kapsamında yürütülmüştür. Çalışma 8 hafta boyunca devam etmiştir. Çalışmada ön test-son test, kontrol gruplu yarı deneysel bir desen kullanılmıştır. Deney grubundaki öğrencilere ders ve laboratuvar saatlerinde görsel programlama öğretimine dayalı öğretim etkinlikleri ile programlama eğitimi verilmiştir. Bunun için bir görsel programlama öğretim aracı olan Blockly yazılımı kullanılmıştır. Kontrol grubundaki öğrenciler ise geleneksel programlama öğretim tarzı ile Python programlama dili üzerinden öğrenim görmüşlerdir.

Çalışmanın başında her iki gruba ön test olarak Kent Üniversitesi Kariyer Danışma birimi tarafından geliştirilen ve araştırmacılar tarafından Türkçe'ye uyarlanan Programlama Yetenek Testi uygulanmıştır. Çalışma sonunda ise her iki gruba son test olarak araştırmacılar tarafından geliştirilen Programlama Başarı Testi ve Başer (2013) tarafından geliştirilen Bilgisayar Programlamaya Karşı Tutum Ölçeği uygulanmıştır. Çalışmadan elde edilen veriler kovaryans analizi (ANCOVA), t-test ve Mann-Whitney U testi kullanılarak analiz edilmiştir.

Çalışma sonucunda görsel programlama öğretimi alan öğrencilerin geleneksel programlama öğretimine göre öğretim alan öğrencilerden programlamayı öğrenmede daha başarılı oldukları tespit edilmiştir. Ayrıca yapılan analiz sonucunda görsel programlama öğretimi ile öğrenim gören öğrencilerin programlamaya karşı tutumları geleneksel programlama öğretimi ile öğrenim gören öğrencilere göre daha yüksek olmasına rağmen aradaki bu farkın istatistiksel olarak anlamlı olmadığı tespit edilmiştir.

Anahtar Sözcükler: programlama, programlama eğitiminin zorlukları, görsel programlama öğretimi, görsel programlama ortamları, Blockly.

ABSTRACT

Student's Name - Surname	Mehmet Fatih YİĞİT
Department's Name	Computer Education And Instructional Technologies
Name of the Supervisor	Prof. Dr. Mustafa BAŞER
Name of the Thesis	Investigating the Effect of Instruction Through Visual Programming Environment on Students' Learning Computer Programming and Attitudes Toward Programming

The aim of this study is to investigate the effect of instruction through visual programming environment on students' understanding of programming and attitudes toward programming. The study was carried out in "Programming Languages I" course with 42 sophomore students from Computer Education and Instructional Technologies department in Ondokuz Mayıs University in the academic year 2015-2016 fall semester. The study lasted 8 weeks. Quasi-experimental pretest-posttest control group design was adopted in this study. Students in the experimental group were taught programming in lectures and laboratory sessions through activities based on visual programming instruction. For this, Blockly, a visual programming tool, was used. Students in the control group were taught programming using Python programming language based on traditional programming instruction

At the beginning of the study, both group was given Computer Programming Aptitude Test, developed by University of Kent Career Advisory Service and translated into Turkish by the researchers. At the end of the study, Computer Programming Achievement Test, developed by the researchers, and Computer Programming Attitude Scale were given both groups. Data obtained by the study were analyzed using analysis of covariance (ANCOVA), t-test and Mann-Whitney U test.

According to the results of the study, the experimental group scored significantly higher than the control group on computer programming achievement test. Moreover, the experimental group developed higher attitude toward computer programming than the control group, but the difference between groups was not statistically significant.

Keywords: programming, learning difficulties in programming, visual programming instruction, visual programming environments, Blockly.

TEŞEKKÜR

Öncelikle yüksek lisans eğitimim ve tez çalışması süreci boyunca maddi ve manevi her konuda yardımlarını esirgemeyen sayın danışman hocam Prof. Dr. Mustafa BAŞER'e teşekkürü bir borç bilirim.

Çalışma süresince deneyimleriyle yardımcı olan Yrd. Doç Dr. Hacı Bayram YILMAZ'a teşekkür ederim.

Çalışma süresince beni motive eden iş arkadaşlarım İsmail ÇETİN, Mustafa Serhat DÜNDAR ve Alpaslan ÜLKEN'e teşekkür ederim.

Çalışmam boyunca destekleri ile her zaman yanımda olan sevgili annem, babam ve kardeşlerime de sonsuz sevgilerimi sunarım.

İÇİNDEKİLER

TABLOLAR LİSTESİ

ŞEKİLLER LİSTESİ

KISALTMALAR LİSTESİ

GPÖ: Görsel Programlama Öğretimi

GEPÖ: Geleneksel Programlama Öğretimi

PYT: Programlama Yetenek Testi

PBT: Programlama Başarı Testi

BPKTÖ: Bilgisayar Programlamaya Karşı Tutum Ölçeği

GPO: Görsel Programlama Ortamı

BÖLÜM 1
GİRİŞ

Programlama özellikle son yıllarda yüksek gelir getiren bir yetenek olarak görülmektedir. Programlama bilmek günümüzde insanlar için iyi bir iş imkanı sağladığı gibi teknoloji merkezli olarak yaşadığımız şu dünyada daha üretken bir ülke haline gelebilmek için de oldukça büyük bir öneme sahiptir (Sak ve Demirer, 2016). Ayrıca programlama öğrenmenin kişiye bilişsel olarak katkı sağladığı da savunulmaktadır (Çakıroğlu, Sarı ve Akkan, 2011; Fessakis, Gouli ve Mavroudi, 2013; Rogozhkina ve Kushnirenko, 2011). Akpınar ve Altun (2014) programlama sayesinde bireyin problem çözme, analitik ve uzamsal düşünme gibi bilişsel becerilerinin geliştiğini ifade ederek programlama öğrenilmesinin önemini vurgulamaktadır. Ancak programlama ile ilgili yapılan araştırmalara bakıldığında programlamanın öğrenilmesi zor bir alan olduğu ve bu durumun da bilgisayar bilimleri alanında yaşanılan en büyük sorunlardan birini teşkil ettiği belirtilmektedir (Ambrosio ve diğerleri, 2011; Bennedsen ve Caspersen, 2005; Gill ve Holton, 2006). Bu sorun, sorunun nedenleri, yol açtığı sonuçlar ve iyileştirilmesi adına yapılan çalışmalar alan yazında sıklıkla ele alınmıştır (ör: Ben-Bassat Levy, Ben-Ari ve Uronen, 2004, Corney, Teague ve Thomas, 2010; Kalelioğlu, 2015; Kinnunen ve Malmi, 2006; Krpan, Mladenović ve Rosić, 2015; Lahtinen, Ala-Mutka ve Jarvinen, 2005; McCracken ve diğerleri, 2001; Rizvi ve Humphries, 2012; Robins, Rountree ve Rountree, 2003).

Programlamanın öğrenilmesi zor bir alan olması çeşitli olumsuz sonuçlar doğurmaktadır. Bunlardan birisi düşük öğrenci notları ve programlama derslerindeki başarısızlıklarıdır. Bununla ilgili olarak Gill ve Holton (2006) yaptıkları araştırmada ilk defa programlama eğitimi alan öğrencilerin yaklaşık % 50'sinin harf notunu D *(en düşük geçme notu)*, W *(dersten çekilme)* veya F *(başarısız)* olarak saptamışlardır. Bennedsen ve Caspersen (2007) çalışmalarında programlamaya giriş dersini alan öğrencilerin % 33'nün dersi geçemediğini belirlemişlerdir. Krpan, Mladenović ve Rosić (2015) 2010-2013 yılları arasında programlamaya giriş dersi alan 511 öğrenci üzerinde yaptıkları çalışmada öğrencilerin %47'sinin programlama dersini başarısızlıkla sonlandırdıklarını tespit etmişlerdir. Teague ve Roe (2009) ise bir üniversitede yaptıkları araştırmada programlamaya giriş dersindeki başarısızlık

oranının ortalama %30 olduğunu belirtmiş ve bu oranın %41'lere kadar çıktığını bildirmiştir.

Buna ek olarak programlama öğretimindeki var olan bu sorundan dolayı, öğrenciler bilgisayar bilimleri programını da bırakabilmektedir. Talton ve diğerleri (2006) ABD'deki Illinois Üniversitesi'ndeki Bilgisayar Bilimleri Programı'nın verilerini dikkate alarak, öğrencilerin % 25'inin ilk yıl sonunda programı bıraktığını açıklamışlardır. Ayrıca programlama öğreniminin zorluğu üniversitelerin Bilgisayar Bilimleri branşını tercih eden öğrenci sayısında da bir azalmaya yol açmaktadır. Zweben (2011) , ABD Bilgisayar Araştırma Birliği (CRA)'nın verilerine göre ABD'de Bilgisayar Bilimlerini tercih eden öğrenci sayısının 2000'den 2007'e kadar % 50 oranında düştüğünü belirtmiştir. Vegso (2005) ise California Üniversitesi'nde yapılan bir çalışmada 2000-2004 yılları arasında Bilgisayar Bilimlerini tercih eden öğrenci sayısında % 60'lık bir düşüşün tespit edildiğini açıklamıştır.

Programlama öğreniminin zorluğu ve bu alanda yaşanılan problemler belli bir ülkeyle sınırlı kalmayıp dünya çapında bir sorun olarak karşımıza çıkmaktadır. Bununla ilgili yapılan uluslararası çalışmalar bulunmaktadır. McCracken ve diğ. (2001) yaptıkları çalışmada birçok ülkeden veri toplamış ve sonuç olarak öğrencilerin programlamaya giriş derslerinden sonra hala programlamayı öğrenemediklerine ve beklenilenden daha düşük bir öğrenci performansıyla karşılaştıklarına dikkat çekmiştir. Araştırma sonunda elde edilen sayısal verilere göre öğrenciler ortalama olarak 110 puan üzerinden 22,89 almıştır. 7 farklı ülkeden veri toplanarak yapılan bir diğer uluslar arası çalışmada ise Lister ve diğ. (2004) öğrencilerin kısa bir program bloğunun ne iş yaptığını saptamakta ve verilmeyen bir kod satırının doğru olarak seçenekler arasından seçilebilme noktasında zorluk yaşadıkları sonucuna ulaşmışlardır. Bunlara ek olarak bilgisayar bilimleri alanında mezuniyet seviyesine gelmiş öğrencileri ele alan ve Amerika, İsveç, Yeni Zelanda gibi ülkelerde yürütülen uluslararası çalışmalarda ise, bu öğrencilerin başarılı bir biçimde program tasarlayamadıkları gözlenmiştir (Eckerdal, McCartney, Moström, Ratcliffe ve Zander, 2006; Loftus, Thomas ve Zander, 2011).

Programlama öğreniminin zor olmasının en önemli sebeplerinden birisi, programlamanın öğrenen kişinin birden fazla tür bilgiye sahip olmasını gerektirmesidir (Mannila, Peltomäki ve Salakoski, 2006). Doğru ve çalışan bir

program yazabilmek için kişi programlama kavramlarını, bunların mantığını, nasıl işlediklerini, bunların programlamaya özgü dille bilgisayarın anlayacağı bir biçimde nasıl yazıldıklarını bilmek ve tüm bunları kullanarak verilen problemin çözümü için uygun bir algoritma geliştirmek durumundadır. Bu süreç programlamaya daha yeni başlayacaklar için oldukça zor ve karmaşık gelmekte ve programlama derslerinde bıkkınlık ve yılgınlık yaşamalarına, sonucunda da başarısız olmalarına sebep olmaktadır (Mannila, Peltomäki ve Salakoski, 2006). Çünkü programlamaya giriş derslerinde tüm bu süreci tam olarak ele almak kolay olmamakta ve bu süreçte programlama için daha fazla önem arz eden bazı becerilerin geliştirilmesi için yeterli zaman bulunamamaktadır. Bundan dolayı alan yazın incelendiğinde de görülmektedir ki programlamaya giriş derslerinde öğrencilerin problem çözme yeteneklerini geliştirmelerine yeterince ağırlık verilmemekte ve daha yoğun olarak programlama söz dizimi ile vakit harcanmaktadır (Jenkins ve Davy, 2000; Lovonen, Meisalo, Lattu ve Suttinen, 2003; McGill ve Volet, 1997). Ancak çeşitli araştırmacılar bunun yanlış olduğunu, programlamaya giriş derslerinde sözdizimi ve kod yazımı ile uğraşmaktan çok programlama kavramlarına ve kod bloğunu okuyup anlamaya dair çalışmalar üzerinde yoğunlaşılmasının öğrenciler açısından daha yararlı olacağını belirtmişlerdir (Lister, Fidge ve Teague, 2009; Lopez, Sutton ve Clear, 2009; Sorva, 2012; Wilson ve Moffat, 2010). Sahip olunması gereken bilgi türünün fazlalığı sebebiyle programlamada yaşanılan zorluğun yanı sıra, programlamaya karşı var olan güdülenme ve tutuma dayalı problemler de programlama öğrenimini zorlaştırabilmektedir (Gomes ve Mendes, 2007; Korkmaz ve Altun, 2013).

Yukarıda bahsedilen bu soruna ilişkin olarak, özellikle programlamaya giriş derslerinde söz dizimini aza indirip problem çözmeyi ön planda tutan, öğrencilerin programlama ile ilgili olarak güdülenmesini artıran ve programlamaya karşı olumlu tutuma sahip olamalarını sağlayan bir takım teknolojilerin kullanılması önerilmiştir (Herbert, 2010, s 4; Powers ve diğ., 2006). Bu teknolojiler daha çok görsel özelliklere sahip olan teknolojiler olarak karşımıza çıkmaktadır. Bunların en bilinenleri ve alan yazında da sıkça bahsedilenleri Scratch (Malan ve Leitner, 2007), Alice'tir (Dann, Cooper ve Pausch, 2000). Bunların ortak özelliği söz diziminden çok öğrencilerin programlama ile ilgili temel kavramları öğrenmelerine ve problem

çözme becerilerini geliştirmelerine ve bunlar üzerinde zaman harcamalarına yardımcı olmasıdır (Myers, 1990).

Scratch ve Alice öğrencilerin programlamaya yönelik temel kavramları öğrenmelerine ve bunun üzerinde daha yoğun bir biçimde çaba sarf etmelerine katkıda bulunmasına rağmen, öğrencilere gerçek bir programlama dili üzerinden öğretim imkanı sunmamaktadır. Bu yazılımlarda öğrenciler sahnede var olan karakter ya da nesneleri program blokları ile hareket ettirmektedirler. Bu yüzden bu teknolojiler üniversite düzeyine değil, daha çok lise ve ilköğretim öğrencilerine hitap etmektedir (Maloney ve diğerleri, 2010). Ancak Google tarafından geliştirilen ve bu tez çalışmasında da kullanılacak olan Blockly yazılımı öğrencilere Scratch ve Alice'teki oyunsu ortam yerine daha gerçekçi ve sözdizimsel programlama dillerinde yapılabilen uygulamalara daha yakın programlar geliştirme fırsatı tanımaktadır. Ayrıca Blockly'de öğrencilerin program blokları ile hazırladıkları kod bloğunun sözdizimsel karşılığına (program koduna) da erişilebilmektedir. Bu özellik Scratch ve Alice'te bulunmamakta ve bu özellik sayesinde programlamanın temel kavramlarını edindikten sonra metin tabanlı programlama dillerine geçiş daha kolay hale gelmektedir (Dann ve diğ., 2012). Bundan dolayı bu çalışmada Scratch ve Alice'in sahip olduğu görsel avantajları ve bunların programlama öğrenmeyi kolaylaştıran özelliklerini barındıran, üniversite seviyesindeki öğrencilere hitap edebilen ve programlamaya giriş derslerinde öğrencilere programlamanın temel kavramlarını öğreterek onları ileride kullanacakları gerçek programlama dillerine hazırlamayı amaçlayan bir yazılım olan Google Blockly görsel programlama ortamı kullanılmıştır. Bu yazılımın programlamaya giriş dersi alan üniversite öğrencilerinin programlama öğrenmelerine ve tutumlarına olan etkisi incelenmiştir.

1.1 Araştırmanın Önemi

Programlamayı öğrenmenin güçlüğü ve programlama öğrenimindeki başarıyı artırmak için önerilen yöntemler alanyazın taramasında bahsedilmiştir. Bu yöntemlerden birisi de görsel programlama ortamlarının kullanılmasıdır. Alanyazında kullanılan görsel programlama ortamları daha çok lise veya daha alt seviyelerde kullanılabilecek oyun tarzı (örn. Scratch, Alice) araçlardır. Bu araçlar gerçek programlama dili ortamı sunmadıklarından, üniversite düzeyindeki öğrencilerde daha etkili araçlara ihtiyaç vardır. Google Blockly, görsel programlama

ara yüzüne sahip ve gerçek programlama dili üzerinden programlamayı öğretmeyi amaçlayan bir yazılımdır. Alanyazında bu aracın programlama öğretimindeki etkisinin incelendiğine dair bir çalışmaya rastlanmamıştır. Bu nedenlerle bir görsel programlama ortamı olan Blockly'nin hem üniversite düzeyindeki öğrencilere hitap etmesi ve gerçek programlama diline ait bir ortam sunması hem de alanyazında bununla ilgili bir çalışmanın olmaması sebebiyle bu çalışmanın alanyazına katkı yapacağı ve bu yüzden de önem taşıdığı düşünülmektedir.

BÖLÜM 2
ALANYAZIN TARAMASI

Bu bölümde programlama eğitimi, programlamayı etkileyen faktörler, programlama öğreniminin zorlukları ve görsel programlama ile ilgili tanımlara, bilgilere ve yapılan çalışmalara yer verilecektir.

2.1 Programlama ve Programlama Eğitimi

Programlama var olan bir problemi çözmek için bilgisayarın anlayabileceği bir dil ile o probleme çözüm üretmektir (Van-Roy ve Haridi, 2004). İlk bilgisayar ile birlikte kullanıcı ve bilgisayar arasındaki iletişimi gerçekleştirebilecek bir aracıya gereksinim duyulmuştur. Bu gereksinim ilk olarak makine dili ile karşılanmaya çalışılmıştır. Ancak makine dilinin zorluğu sebebiyle daha kolay yazılabilen ve anlaşılabilen programlama dilleri geliştirilmiştir. Programlama dili, programcının bilgisayardan ne yapmasını istediğini anlatabilmek için kullandığı sözdizimleridir (Van-Roy ve Haridi, 2004). Günümüzdeki en popüler ve en yaygın olarak kullanılan programlama dilleri Python (Nikula, Sajaniemi, Tedre ve Wray, 2007), Java, C ve C++ 'tır (Moons ve De Backer, 2013).

Bir bilgisayar programının meydana getirilmesinde ise bir takım aşamalar bulunmaktadır. Casey (1997) bu aşamaları problemi kavrama, kod tasarlama, olası hataları ayıklama ve programın doğru çalıştığını teyit etme olarak sıralamıştır. Diğer bir ifadeyle, programcının ilk olarak yapması gereken problemi iyi bir şekilde anlamak ve analiz etmektir. Daha sonraki aşamada ise programcı var olan bu problemin çözümü için gerekli olan kodu tasarlamak durumundadır. Programcı kod tasarımından sonra programda var olan hataları gidermek veya programı daha etkin bir şekilde çalışır hale getirmek için programı revize etmeye ihtiyaç duymaktadır. Son aşamada ise doğru, sorunsuz ve istendik bir biçimde çalıştığı teyit edilir.

Yukarıda da belirtildiği gibi bir programcı kod yazımı süresince birden fazla çeşit bilgi ve beceriye sahip olmak durumundadır. Bunlar şu şekilde özetlenebilir (Bayman ve Mayer, 1988; Linn, 1985) :

1- Sözdizimsel (Syntactic) Bilgi: Bir programlama dilinin kullanımı için gerekli olan yazım biçimine ait bilgi. Python programlama dilinden örnek vermek gerekirse,

işlev, döngü ve karşılaştırmaların içerisindeki kod bloklarının satır başı yapılarak yazılmaya başlanması gerektiğini ya da liste ya da sözlükler için hangi tür parantezin ([ ], {}) kullanılması gerektiğini bilmek sözdizimsel bilgiye girmektedir. Program yazabilmek için sözdizimsel bilgi kendi başına yeterli değildir. Bunların anlamlı ve problemin çözümüne uygun bir şekilde tasarlanması gerekmektedir. Bunun içinde aşağıda bahsedilecek bilgi türlerine de sahip olmak gerekmektedir (Bayman & Mayer, 1988).

2- Kavramsal (Conceptual) Bilgi: Programlamada yer alan kavramlara ait bilgi. Bu bilgi türü programlama dilleri arasında pek fazla farklılık göstermemekte bunlar arasında rahat bir şekilde transfer edilebilmektedir. Hemen hemen her programlama dilinde döngü mantığı ve ne iş yaptığı benzerlik göstermektedir. Örneğin "for" döngüsü belirtilen koşul sağlanana kadar içerisinde yazılan kodu işletmeye devam eden bir döngü biçimidir.

3- Stratejik (Strategic) Bilgi: Programlamada verilen problemin çözümü için gerekli olan problem çözme becerisi. Diğer bir deyişle, program yazmak için sahip olunması gereken sözdizimsel ve kavramsal bilgiyi kullanarak problemin çözümüne yönelik etkin bir algoritma geliştirilme sürecinde büyük öneme sahip bilgi türüdür. Örneğin verilen bir sayının faktöriyelini bulan bir program yazılması istendiğinde, kişi bunun için kullanması gereken işlev ya da döngü kavramlarını, bunların mantığını ve sözdizimsel olarak nasıl yazılması gerektiğini bilebilir. Ancak bu kavramları kullanarak nasıl bir algoritma izlemesi gerektiğini de bilmesi gerekir ve bu süreç kişinin stratejik bilgisi ile ilgilidir. Bu 3 tip bilginin daha iyi anlaşılabilmesi için programlamanın bir dilde şiir yazmaya benzetildiği düşünüldüğünde, kişinin öncelikli olarak dilbilgisini bilmesi (sözdizimsel bilgi), daha sonra kelimeleri ve anlamlarını bilmesi (kavramsal bilgi) ve son olarak da bir duyguyu ifade edecek şekilde kelimeleri yan yana getirmeyi bilmesi (stratejik bilgi) gerekmektedir.

2.2 Acemi Programcılar

Acemi programcı daha önce bir programlama öğretimi görmemiş ve programlama ile ilgili bir alt yapısı olmayan bireylerdir (Mow, 2012). Acemi programcılar hakkında yapılan çalışmalar, programlamada düştükleri hataları ve yaşadıkları güçlükleri tanımlamak açısından önemlidir (Ebrahimi, 1994). Çünkü bu gibi problemleri tanımlamadan, bunların giderilmesi için gerekli olan programlama öğretim

stratejilerinin geliştirilmesi zordur. Acemi programcılar programlama eğitimlerine birçok engellerle karşılaşarak başlamaktadırlar. Bati, Gelderblom ve van Biljon (2014) bu engelleri kavram yanılgıları, akademik geçmişlerinin yetersizliği, programlama ile ilgili kavramlara yabancılık çekme ve güdülenme (motivasyon) olarak belirtmişlerdir.

Literatürde ayrıca acemi programcı olarak nitelendirilen bireylerin birçok özelliği olduğu tespit edilmiştir. Örneğin çoğu acemi programcı ya hiçbir programsal alt yapıya sahip değil (Mow, 2012), ya da sadece programlamaya giriş derslerinde eğitim görmüşlerdir (Madison ve Gifford, 2002; Gobil, Shukor ve Mohtar, 2009). Ayrıca acemi programcılar yüzeysel bir programlama bilgisine sahip olmakta (Winslow, 1996), bildiklerini programa yansıtamamakta ve sözdizimsel ve kavramsal bilgiye sahip olsalar dahi, bunları anlamlı ve tam bir program haline getirmekte güçlük çekmektedirler (Kelleher ve Pausch, 2005; Lahtinen, Ala-Mutka ve Jarvinen, 2005). Bunlara ek olarak, acemi programcılar problem çözme ve planlama becerisinden yoksun olmakta ve bunlar üzerinde az zaman harcamaktadırlar (Lane ve VanLehn, 2005; Mhashi ve Alakeel, 2013; Soloway ve Ehrlich, 1984).

2.3 Programlama Öğrenimini Etkileyen Faktörler

Her öğrenci programlamayı eşit düzeyde öğrenememektedir. Bazı öğrenciler programlamayı daha çabuk ve kolay öğrenirken, bazıları da yavaş ve geç öğrenebilmektedir (Wray, 2007). Bu durum alanyazında da incelendiği üzere programlama öğrenimini etkileyen bir takım faktörlerin bulunduğunu göstermektedir (Bergin ve Reilly, 2006; Byrne ve Lyons, 2001; Caspersen, Larsen, Bennedsen, 2007). Bunlardan en sıkça bahsedilenleri, geçmiş programlama tecrübesi (Rountree, Rountree, Robins ve Hannah, 2004), programlama öz-yeterliği (Mazman, 2013), matematik bilgisi (Ventura, 2005), problem çözme becerisidir (Pillay ve Jugoo, 2005).

Geçmişte elde edilen programlama tecrübesi programlama öğrenimini etkileyen en önemli faktörlerden biri olarak gösterilmektedir (Fincher ve diğerleri, 2006; Rountree ve diğerleri, 2004). Programlama ile ilgili olarak önceden elde edilmiş bir takım tecrübeler programlama öğrenimi kolaylaştırmakta ve öğrencilerin daha başarılı olmalarını sağlamaktadır. Bununla ilgili olarak yapılan çalışmalara

bakıldığında lisans düzeyinden önce elde edilen programlama tecrübelerinin üniversitede programlamaya giriş derslerindeki başarıya olumlu katkı sağladığı gözlemlenmiştir (Hagan ve Markham, 2000; Wiedenbeck, Labelle ve Kain, 2004).

Programlama öz-yeterliği de programlama öğrenimini etkileyen bir diğer faktördür. Öz-yeterlik bireyin bir görevi başarıp başaramayacağına ilişkin inancıdır (Bandura, 1977). Bu bağlamda bireyin programlama hakkındaki öz-yeterlik algı düzeyi programlama başarısını olumlu ya da olumsuz etkileyebilmektedir. Bu konuda yapılan çalışmalara bakıldığında öz-yeterliğin programlama başarısını yordadığı ve programlama başarısı ile anlamlı bir ilişkiye sahip olduğu görülebilmektedir (Mazman, 2013; Ramalingam, LaBelle ve Wiedenbeck, 2004; Wiedenbeck, LaBelle ve Kain, 2004).

Programlama öğreniminde ayrıca bireyin matematik altyapısının da etkili olduğu belirtilmektedir (Byrne ve Lyons, 2001; Wilson ve Shrock, 2001). Ventura (2005) matematiksel beceri ile programlama başarısını birlikte ele alan çalışmalardan yola çıkarak matematik bilgisinin programlama başarısı ile anlamlı bir ilişkisi olduğunu ifade etmektedir. Yapılan bazı çalışmalar incelendiğinde de matematiksel becerileri düşük olan öğrencilerin programlama derslerinde de başarısız oldukları görülmektedir (Byrne ve Lyons, 2001; Gomes ve diğ., 2006).

Programlama öğreniminde bir diğer etkili faktör ise problem çözme becerisidir (Pillay ve Jugoo, 2005; Yurdagül ve Aşkar, 2013). Gomes ve Mendes (2007) problem çözme becerisinin programlama öğreniminde oldukça önemli olduğunu ve programlama öğreniminde yaşanılan zorlukların çoğunun problem çözme becerisi eksikliğinden kaynaklandığını belirmektedir. Ayrıca problem çözme becerisinin programlama başarısını etkisini inceleyen çalışmalar bakıldığında ise bu ikisi arasında anlamlı bir ilişki olduğu tespit edilmiştir (Pillay ve Jugoo, 2005; Yurdagül ve Aşkar, 2013).

Yukarıda da açıklandığı üzere programlama öğrenme sürecini etkileyen bir takım faktörler bulunmaktadır. Bu faktörler ayrıca öğrencilerin programlama başarısını önceden tahmin edebilmek için kullanılan programlama yetenek testleri için birer yardımcı durumundadır. Programlama yetenek testleri öğrencilerin ya da kurum çalışanlarının programlamada başarılı olup olamayacaklarını yordamak için ve programlama konusunda potansiyel sahibi olduğuna inanılan bireylerin seçimi için

kullanılan testlerdir (Tukiainen ve Mönkkönen, 2002). Bu testlerin en bilinenleri IBM programlama yetenek testi (Pea ve Kurland, 1983), Dehnadi testi (Dehnadi, 2006) ve Kent Üniversitesi programlama yetenek testidir (Computer Programming Aptitude Test, 2015). Programlama yetenek testleri programlama ile ilgisi olan ve yukarıda belirtildiği üzere programlama öğrenimine etkisi olan matematik bilgisi ve problem çözme becerisi gibi bir takım faktörlerin ölçümünde kullanılmakta ve elde edilen sonuçlara göre bireyin programlamada başarılı olup olamayacağı konusunda bazı tahminler ortaya koymaktadır (Lacher ve Lewis, 2015; Tukiainen ve Mönkkönen, 2002).

Programlama yetenek testleri ile öğrencilerin programlama başarısı arasındaki ilişkiyi inceleyen az da olsa bazı çalışmalar mevcuttur. Harris (2014) kendi geliştirmiş olduğu PATT programlama yetenek testi ile programlama başarısı arasında anlamlı bir ilişki bulduğunu belirtmiştir. Lacher ve Lewis (2015) Kent Üniversitesi tarafından geliştirilen programlama yetenek testini kullanmış ve öğrencilerin programlama başarısı ile yetenek testi puanları arasında pozitif bir korelasyon tespit etmiştir. Dehnadi (2006) ise yaptığı çalışmada geliştirmiş olduğu programlama yetenek testinin programlama başarısını tahmin etmede etkili olduğundan bahsetmiş ancak daha sonra yaptığı çalışmada testin tahmin yeteneğinin ilk çalışma kadar güçlü olmadığını belirtmiştir (Bornat ve Dehnadi, 2008).

2.4 Programlama Öğrenmenin Zorlukları

Programlama öğreniminin zor olmasının en başta gelen sebeplerinden birisi, programlamayı öğrenecek kişinin birden fazla tür bilgiye sahip olmasının gerekmesidir (Mannila, Peltomäki ve Salakoski, 2006). Programlama öğrenmeye başlayan bir kişi sadece verilen problemi çözme ile uğraşmamakta, buna ek olarak programa ait sözdizimsel ve kavramsal bilgiye de sahip olmak durumunda kalmaktadırlar (Mannila, Peltomäki ve Salakoski, 2006). Bu ise programlama eğitimini bir kat daha fazla zor hale getirmektedir.

Edirisinghe (2008) ve Cooper, Dann ve Pausch (2000a) yeni öğrenenler için programlamayı zorlaştıran sebepler arasında, yazılan kodun kavranması ve analiz edilmesi için gerekli olan becerilerin yetersizliğini öne sürmektedir. Bu tür becerilerden yoksun olmak kod yazımı sırasında yapılan hataları saptamayı ve sonrasında da düzeltmeyi zor bir hale getirmektedir. Bu tür öğrenciler hatayı tam

olarak belirleyemediklerinden hatayı gidermek için bilinçli bir şekilde değil rastgele değişikliklere başvurmaktadırlar (Polo, 2013). Bunun sonucunda da, hatayı düzeltseler dahi hatanın nasıl ve niçin düzeldiğini tam olarak anlayamamakta ve ileride de bu hataları tekrarlamaktadırlar (Edirisinghe, 2008).

Programlama öğreniminin zor olmasının bir diğer sebebi ise, programlamanın öğrenciler için yeni öğrenilecek bir ders olmasıdır (Jenkins, 2002). Öğrenciler matematik, Türkçe ve fen bilimleri gibi dersler ile ilgili kavramlarla okul yaşamlarının hemen başında ve hatta okul öncesinde de karşılaşmaktadırlar. Bundan dolayı öğrenciler bu derslere yabancılık çekmemektedirler. Bunun aksine öğrenciler programlamaya giriş ile ilgili temel kavramlara küçük yaşlarda değil, genellikle lisans seviyesinde tanışabilmektedirler (Zaharija, Mladenović ve Boljat, 2013). Bu da programlama derslerini diğer derslere göre daha zor kılmakta ve öğrencilerin daha başarısız olmasına sebep olmaktadır.

Güdülenme eksikliği de programlama dersini zorlaştırmaktadır. Yapılan çeşitli araştırmalarda akademik başarı ile güdülenme arasında pozitif bir ilişki olduğu bulunmuştur (Eymur ve Geban, 2011; Goodman ve diğerleri, 2011; Tella, 2007). Dolayısıyla güdülenme ile ilgili problemler akademik başarıyı olumsuz etkileyebilmektedir. Gomes ve Mendes (2007) programlama derslerinde var olan bu güdülenme eksikliğini dersi alan öğrencilerin yeni öğrencilere programlama dersinin devamlı olarak zorluklarından bahsederek bu öğrenciler üzerinde "programlama dersi zordur" ön yargısını oluşturmalarına bağlamaktadır. Yine bununla ilgili olarak öğrencilerin programlamaya karşı sahip oldukları öz-yeterlik algısındaki yaşanılan bir takım problemler de programlama başarısını olumsuz etkilemektedir (Korkmaz, 2013).

Ayrıca programlamaya giriş derslerinde kullanılan programlama türü de tartışma konusudur, özellikle acemi öğrencilerin programlamayı öğrenmelerinde etkilidir (Goldwasser ve Letscher, 2008). Bazı araştırmacılar programlamaya giriş derslerinde nesne tabanlı programlama kullanımının, öğrencilerin temel programlama kavramlarını öğrenmelerinin yanı sıra sınıf ve nesne gibi daha karmaşık kavramları da öğrenmelerini gerektirdiğini ve bunun da programlama öğrenimini güçleştirdiğini ve öğrencilere zihinsel olarak daha fazla yüklendiğini savunmaktadır (Cooper, Dann ve Pausch, 2003; Proulx, Raab ve Rasala, 2002). Bundan dolayı nesne tabanlı

programlamada öğrenilmesi gereken kavramların sayısı arttığından, her bir kavram daha hızlı ve daha yüzeysel olarak işlenmekte ve döngüler, karşılaştırmalar gibi daha temel kavramların önemine çok fazla vurgu yapılamamakta, öğrencilerin programlama ile ilgili temelleri zayıf bir biçimde atılmakta ve ileride daha fazla sorunla karşılaşmaktadırlar (Bruce, 2005; Hu, 2004; Scott, 2010). Bununla ilgili olarak yapılan bir araştırmada, nesne tabanlı programlamadan işleve dayalı programlamaya yapılan geçişin, öğrenci başarısını, memnuniyetini ve öğrencilerin derse olan katılımlarını arttırdığı gözlenmiştir (Reges, 2006).

Programlamaya karşı tutumun da programlama öğreniminde etkisini söylemek mümkündür. Programlamaya karşı sahip olunan olumsuz tutumun programlamayı öğrenme sürecini olumsuz bir biçimde etkileyebileceğini ve zorlaştırabileceğini belirten çalışmalar bulunmaktadır (Anastasiadou ve Karakos, 2011; Farkas ve Murthy, 2005; Korkmaz ve Altun, 2013; Özyurt ve Özyurt, 2015). Çünkü alanyazında da görüleceği üzere, bu görüşü destekleyen ve tutum ile akademik başarı arasında pozitif bir ilişki olduğunu gösteren birçok çalışma bulunmaktadır (Başer, 2013; Ghaith, 2003; Golding, Facey-Shaw ve Tennant, 2006; Martinez, Aricak ve Jewell, 2008; Sorge ve Schau, 2002; Tai ve diğerleri, 2003). Bu yüzden öğrencilerin programlamaya karşı olumlu tutum sergilemelerini sağlamak, programlama öğrenimini daha kolay hale getiren bir yol olarak değerlendirilebilir (Korkmaz ve Altun, 2013).

Acemi programcıların özellikleri ve yukarıda da belirtildiği üzere alanyazında geçen programlama ile ilgili zorluklar göz önünde bulundurulduğunda, programlama eğitimi ile ilgili bir takım düzenlemelere ihtiyaç olduğu görülmektedir. Aksi takdirde öğrenciler programlama ile ilgili olumsuz tutuma sahip olmakta, isteksizlik ve güdülenme eksikliği yaşamakta ve başarısız olarak programlama derslerini tamamlamaktadırlar (Bishop-Clark, Courte, Evans ve Howard, 2007). Bundan dolayı programlamayı öğrenmeyi kolaylaştırmaya yönelik bazı adımların atılması gerekmektedir. Herbert (2010) bu adımları programlamadaki sözdizimleri en aza indirmek ve görsel araçlardan yararlanmak şeklinde belirtmiştir. Adams (2014) ise öğrencilerin ilgilerini çekecek ve güdülenmesini arttıracak teknolojilerin kullanılmasının programlama öğrenimindeki sorunları giderebileceğine değinmektedir.

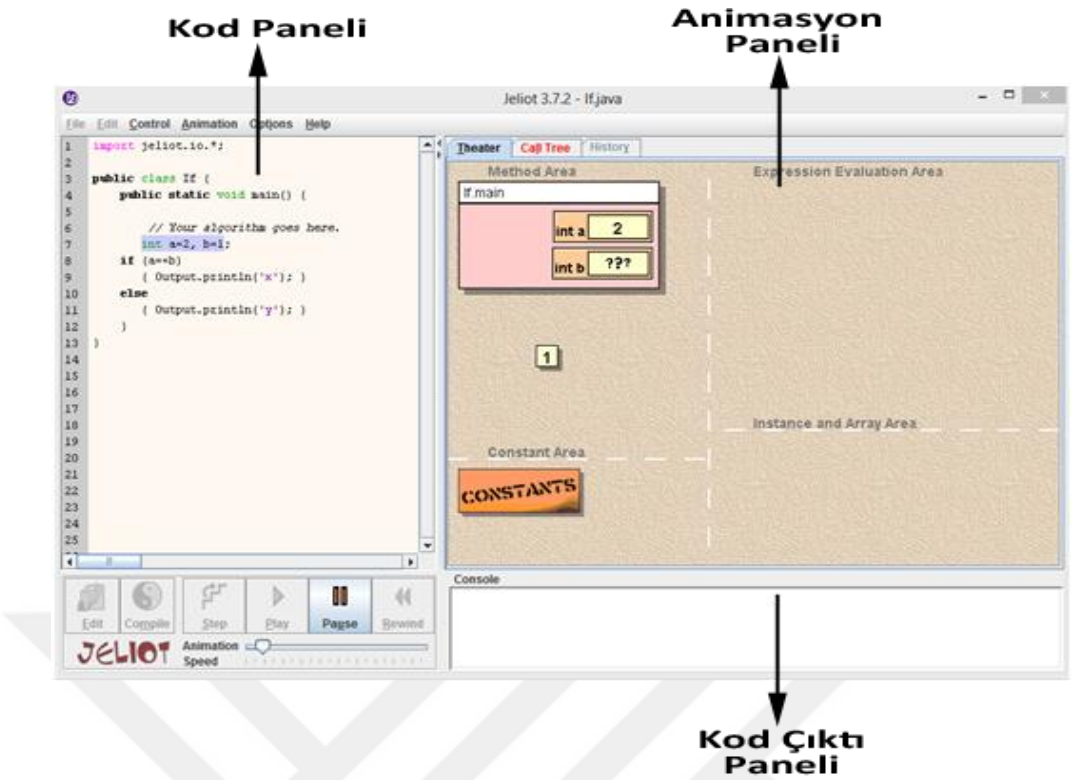
2.5 Programlama Eğitiminde Görsel Teknolojiler

Alanyazında da sıkça bahsedildiği üzere programlama öğrenimi zor bir süreçtir. Bu zorluk programlamaya giriş derslerinde kendisini daha fazla hissettirmektedir. Bu sorunun üstesinden gelebilmek için programlamayı oyun üzerinden öğrenme (Maragos ve Grigoriadou, 2011; Muratet, Torguet, Viallet ve Jessel, 2011; Paliokas, Arapidis ve Mpimpitsos, 2011) veya eşli olarak *(pair programming)* öğrenme (Müller, 2007; Šerbec, Kaučič ve Rugelj, 2008; Williams ve diğerleri, 2002) gibi farklı metotlar bulunmaktadır. Bunlara ek olarak programlama öğreniminde görsel teknoloji kullanımı da tercih edilen stratejilerden birisidir (Malan ve Leitner, 2007). Görsel teknoloji program oluştururken ya da program sonrasında görselleştirmelerden yararlanan teknolojilerdir (Laakso, Kaila ve Salakoski, 2008; Meerbaum-Salant, Armoni ve Ben-Ari, 2011) Görsel teknolojinin programlama öğretimi sürecine dahil edilmesinin altında bir takım nedenler yatmaktadır. Kaucic ve Asic (2011) programlama öğretiminde görsel teknolojilerin kullanılması ile ilgili yapılan araştırmaların belli ortak sonuçlara ulaştıklarına değinmişlerdir. Buna göre bu gibi teknolojilerden yararlanmanın öğrenci güdülenmesine ve öğrencilerin programlamaya karşı olumlu tutum sergilemeleri noktasında pozitif katkı sağladığı görülmektedir. Naps ve diğerleri (2002) görsel teknolojilerin programlamanın soyutluğunu daha somut bir hale getirerek, programlamayı öğrenmeyi kolaylaştırdığını belirtmişlerdir. Shu (1999) da görsel teknoloji kullanımının bilişsel bir biçimde programlamaya katkı sağladığını savunarak, görsellerin daha fazla anlam taşıdığını, anlamayı ve hatırlamayı kolay hale getirdiğini ve programlamayı daha eğlenceli ve ilgi çekici kıldığından bahsetmiştir.

Programlama öğreniminde görsel teknoloji kullanımının nedenlerinden bir diğeri de, öğrencilerin görsel öğrenme stiline sahip olduklarının görülmesidir (Allert, 2004). Bunu destekleyen birçok çalışma bulunmaktadır (Chen ve Lin, 2011; Fowler ve diğ., 2000; Gomes ve Mendes, 2010; Layman, Cornwell ve Williams, 2006; Thomas ve diğ., 2002; Zualkernan, Allert ve Qadah, 2006). Görsel öğrenenler resim, grafik, diyagram, animasyon gibi araçlarla daha kolay öğrenmekte ve öğrendiklerini de daha rahat bir şekilde hatırlayabilmektedirler (Sims ve Sims, 1995). Ayrıca ikili kodlama teorisine (Paivio, 1990) göre de metin ile birlikte görsellerin kullanılması bilgiyi kodlamayı kolaylaştırmakta ve öğretimsel hedeflere ulaşılmasını daha mümkün hale getirmektedir (Clark ve Lyons, 2010). Görsel teknolojilerin yararları ve öğrencilerin

de öğrenme stilleri göz önünde bulundurulduğunda, bu tip teknolojilerin programlama öğretiminde kullanılması gerektiği daha net bir biçimde anlaşılmaktadır. Myers (1990) programlama alanında kullanılan görsel teknolojileri program görselleştirme araçları ve görsel programlama ortamları olarak ikiye ayırmıştır.

Program görselleştirme araçları (PGA) yazılan programın hangi aşamalardan geçerek çalıştığını görsel bir animasyon olarak sunan teknolojilerdir. PGA'nın temel amacı özellikle programlamaya yeni başlayanlara, programın işleyiş sürecinin ve programlama mantığının görselleştirilerek daha rahat bir biçimde anlatılabilmesidir (Laakso, Kaila ve Salakoski, 2008). PGA'nın bunu amaç edinmesinin altında yatan en büyük nedenlerden birisi, çoğunlukla programlamaya yeni adım atanların sorun yaşadığı programı kavramanın ve işleyişini anlayabilmenin programlamadaki sahip olunması gereken en önemli becerilerden biri olmasıdır (Kirby, Toland ve Deegan, 2010; Moreno, Myeller ve Bednarik, 2005). Böylelikle, PGA yeni öğrenenlere yaşadıkları bu sıkıntı noktasında faydalı olabilecek bir görsel teknoloji olarak karşımıza çıkmaktadır (Sorva, Lönnberg ve Malmi, 2013). Son yıllarda bu alanda kullanılan yazılımlardan en bilineni Jeliot yazılımıdır. Jeliot Java programlama dili üzerinde görselleştirme yapmaktadır. Şekil 2.1'de de görüldüğü üzere Jeliot üç ana panelden oluşmaktadır. Soldaki panelde program kodları yer almaktadır. Kullanıcı Java kodunu bu panele girerek programı çalıştırmakta ve bir animasyon halinde programın adım adım olarak işleyişi sağdaki panelde takip edilebilmektedir. Alt panelde ise programın çıktısı gösterilmektedir.



Şekil 2.1: Jeliot ara yüzü

Programlamaya yeni başlayan öğrenciler programın işleyiş süreci ve bu süreçte olanlar hakkında yetersiz bilgiye sahip olmakta ve bu süreç onlara soyut gelmektedir (Kirby, Toland ve Deegan, 2010). Jeliot sahip olduğu görsel özellikler sayesinde bu süreci somutlaştırarak, programın hangi adımında ne gibi işlemlerin gerçekleştiği noktasında kullanıcılara önemli kolaylıklar sağlamaktadır (Kannusmäki, Moreno, Myller ve Sutinen, 2004). Jeliot giriş-çıkış işlemleri, diziler, operatör ve atamalar, kontrol yapıları, döngüler ve fonksiyonlar gibi temel programlama kavramlarını görselleştirebilmektedir (Moreno ve Myller, 2003).

Alanyazında Jeliot yazılımının programlama öğrenimindeki etkilerini inceleyen araştırmalara bakıldığında, Jeliot'un döngü ve karşılaştırma gibi temel programlama kavramlarının öğrenciler tarafından anlaşılmasını kolaylaştırdığı, yazılan programın hangi aşamalarda ne tür değişikliklere uğradığı (Kannusmäki, Moreno, Myller & Sutinen, 2004) ve program çıktısını tahmin etme uygulamalarında öğrenci başarısını artırdığı gözlemlenmiştir (Ben-Bassat Levy, Ben-Ari ve Uronen, 2004).

2.5.1 Görsel Programlama Ortamları

Programlama alanındaki öğrenme zorluklarını giderebilmek için kullanılan yöntemlerden birisi görsel programlama ortamlarıdır (GPO) (Malan ve Leitner, 2007). GPO kullanıcıların söz dizimsel olarak bir program oluşturmalarının aksine, görsel öğeler aracılığıyla programlama yapmalarını temel almaktadır (Kaucic ve Asic, 2011; Shu, 1999). Bu tanımda da belirtildiği üzere, GPO'nun en belirgin özelliği, kullanıcıların söz diziminden ziyade programlama ile ilgili temel kavramları önemsemesine ve üzerinde daha fazla zaman harcamalarını sağlamasıdır (Myers, 1990). Çünkü, öğrenciler programlama öğrenmeye başlarken öncelikle temel kavramları değil söz dizimini öğrenmeye çalışmaktadırlar (Jenkins ve Davy, 2000; Lovonen, Meisalo, Lattu ve Suttinen, 2003; McGill ve Volet, 1997). Wilson ve Moffat (2010) bunun yanlış olduğunu, öğrencilerin temel kavramlara öncelik vermesi gerektiğini ve GPO'nun da bu noktada oldukça etkili olduğunu belirtmektedir.

Programlamaya giriş derslerinde kod yazmaktan çok kod bloğunu okuyup anlamanın, verilen kodun ne tür aşamalardan geçerek çalıştığının ve kod çalıştıktan sonraki çıktının ne olacağına ilişkin aktivitelerin öğrenciler için daha yararlı olacağı çeşitli araştırmacılarca belirtilmiştir (Lister, 2001; Lister ve diğerleri, 2004; Lopez, Sutton ve Clear, 2009; Sorva, 2012). Diğer bir deyişle, programı okuyup kavramanın program yazmanın ön koşulu olduğu ve programlamaya giriş derslerinde daha öncelikli işlenmesi vurgulanmıştır. GPO'nun da program yazmadan ziyade programlama kavramlarına önem verdiği göz önüne alındığında alanyazında var olan bu görüşe ayak uydurduğu görülebilmektedir.

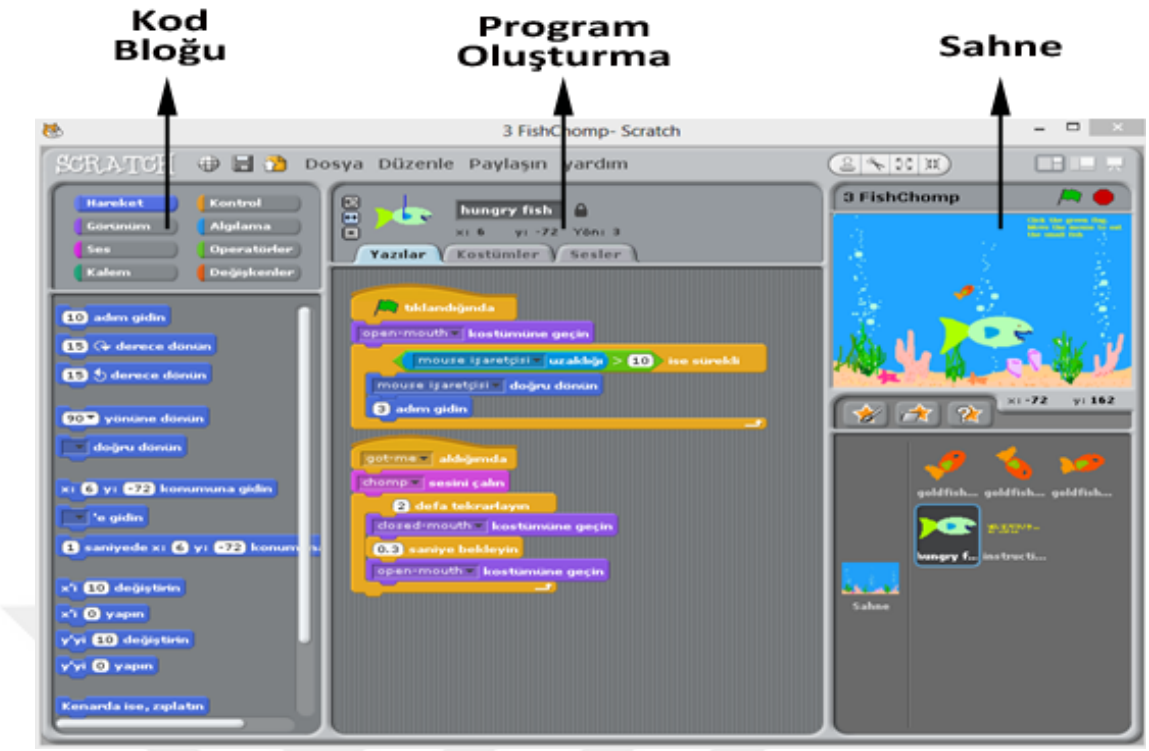
Görsel programlama alanında yapılan çalışmalara bakıldığında, ilk olarak LOGO göze çarpmaktadır (Papert, 1980). LOGO temel olarak ekranda yer alan bir kaplumbağanın kullanıcıların girmiş oldukları komutlara göre hareket ederek çeşitli şekiller meydana getirmesi amacına yönelik bir biçimde tasarlanmıştır. Günümüze gelindiğinde ise, yeni öğrenenlere programlamayı öğretmek için kullanılan görsel programlama ortamları arasında en bilinenleri Scratch, Alice ve Blockly'dir (Powers ve diğerleri, 2006). Bu görsel programlama ortamlarının en öne çıkan ortak özelliği programların blokların birleştirilmesi sonucu oluşturulmasıdır. Her bir blok bir programlama kavramına ya da parçasına denk gelmektedir. Bunların birleştirilmesi

sonucunda ise kullanıcılar programlarını oluşturabilmektedirler. Weintrop ve Wilensky (2015) blok kullanımının bazı avantajları olduğundan bahsetmiş ve yaptıkları araştırmada öğrencilerin blokların günlük kullanılan dile daha yakın olduğunu düşündükleri ve böylelikle programlama kavramlarını daha rahat anlayabildikleri sonucuna ulaşmışlardır.

2.5.1.1 Scratch

Scratch çoğu görsel programlama ortamında olduğu gibi kod yazmanın aksine kod bloklarını sürükle bırak mantığı çerçevesinde kullanarak program oluşturmaya yarayan bir yazılımdır (Meerbaum-Salant, Armoni ve Ben-Ari, 2011). Scratch'ın temel amacı programlamayı yeni öğrenmeye başlayanlara, söz diziminden önce daha gerekli olan programlama ile ilgili temel kavramları ve programlama mantığını öğretebilmektir (Malan ve Leitner, 2007). Diğer bir deyişle, Scratch yeni öğrenenleri söz diziminin katı ve karmaşık yapısından kurtararak onları daha çok problemin çözümü ve programlamanın temelini oluşturan operatörler, karşılaştırmalar, döngüler ve fonksiyonlar gibi kavramlar üzerinde yoğunlaşmalarını ve zaman harcamalarını sağlamaktadır (Kaucic ve Asic, 2011).

Şekil 2.2'de görüldüğü üzere Scratch programında dört ana alan bulunmaktadır. Bunlar kod blokları alanı, program oluşturma alanı, sahne ve karakter alanıdır. Kullanıcılar öncelikle sahnede kullanacakları karakterleri belirleyip daha sonra kod bloklarını program oluşturma alanına sürükleyip bırakarak program oluşturmakta ve oluşturulan programın sonucunda da sahnedeki karakterlerde oluşan değişiklikleri anında gözleyebilme fırsatını yakalamaktadırlar.



Şekil 2.2: Scratch arayüzü

Scratch'ta kodlar hazır olarak bloklar halinde olduklarından, kullanıcıların söz dizimsel hatalara düşme olasılığı bulunmamaktadır. Ayrıca blokların şekilleri onların birbirleriyle nasıl birleştirileceği hakkında ipucu vermektedir. Dolayısıyla kullanıcıların birbirleriyle anlamsız bir şekilde blokları birleştirmelerine program izin vermemekte ve bu sayede kullanıcılar kavramlar arasındaki ilişkileri daha rahat anlayabilmektedirler. Buradan Scratch'ın programlama öğrenme sürecine önemli katkılar sağlayabileceği söylenebilir. Kordaki (2012) Scratch'ın avantajlarını şu şekilde sıralamaktadır:

- Aktif ve etkileşimli öğrenme ortamı sunması,
- Söz dizimden çok programlama mantığına önem vermesi,
- Problem çözümünü desteklemesi,
- Program ve çıktısını görsel hale dönüştürmesi,
- Anında geribildirim vermesi,
- Öğrenenin güdülenmesini artırması.

Rizvi ve Humphries (2012) yaptıkları çalışmada Scratch'ın programlamayı öğrenmedeki etkilerini incelemişlerdir. Uygulamayı yaptıkları üniversitede

programlamaya giriş dersini (CS1) alabilmek için belli bir matematik alt yapısına sahip olmak gerekmektedir. Yerleştirme sınavında düşük matematik puanı alan öğrenciler CS1'in öncesinde bir hazırlık dersi olarak belirtilen CS0 dersini almak zorundadır. CS0 dersi Scratch üzerinden işlenmektedir. Matematik puanı yeterli görülen öğrenciler ise CS1 dersini doğrudan alabilmektedir. Bu iki grubun CS1 dersini geçme yüzdeleri karşılaştırıldığında, önceden Scratch ile CS0 alan grubun %74'ü CS1 dersini geçerken, bu oran diğer grupta %39'da kalmaktadır.

Fesakis ve Serafeim (2009) Scratch'ın öğrencilerin programlamaya karşı tutumlarına etksini incelemiştir. Agean Üniversitesi'nde 35 öğrenci üzerinde yapmış olduğu çalışmada Scratch kullanımının öğrenicilerin programlama dersi hakkındaki tutumlarını olumlu olarak etkildiğini belirtmiştir.

Kalelioğlu ve Gülbahar (2014) 49 ilköğretim düzeyindeki öğrenciye Scratch yazılımını kullandırmıştır. Çalışmanın sonunda Scratch'ın öğrencilerin programlamayı sevmelerinde etkili olduğunu bulgulamıştır.

Shin ve Park (2014) ilkokul öğrencileri ile ilgili olarak Scratch'ın problem çözme becerisine etkisini araştırmıştır. 6. sınıf düzeyindeki 46 öğrenci ile yaptıkları çalışma sonucunda öğrencilerin problem çözme becerilerinin Scratch ile geliştiğini gözlemlemişlerdir. Buna benzer bir sonuç elde eden başka bir çalışmada da Scratch'ın problem çözme becerisi üzerinde önemli bir etkisinin olduğu tespit edilmiştir (Brown ve diğerleri, 2013).

Malan ve Leitner (2007) üniversite öğrencilerinin Scratch hakkındaki görüşlerini ele alan bir çalışma yürütmüşlerdir. Katılımcıların %76'sı Scratch'ın kendileri üzerinde olumlu etkiye sahip olduklarını belirtmişlerdir. Bu öğrenciler çoğunlukla daha önceden programlama tecrübesi olmayan öğrencilerdir. Scratch'ın herhangi bir etkisi olmadığını düşünen öğrencilerin (%16) ise tamamı programlama tecrübesi bulunan öğrencilerdir.

Ozoron, Çağıltay ve Topallia (2012) bilgisayara giriş dersine Scratch yazılımını entegre ederek Scratch ortamının etkilerini araştıran bir çalışma yapmışlardır. Araştırma sonunda Scratch kullanımının :

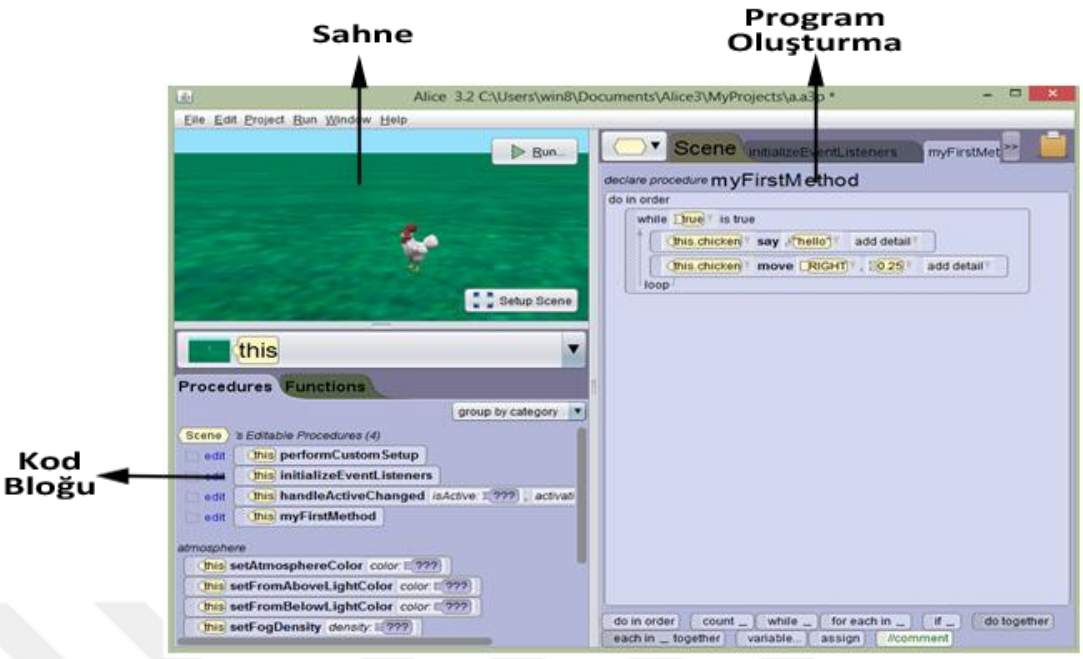
- Programlamayı daha eğlenceli ve görsel hale getirdiği,
- Algoritma öğrenimini kolaylaştırdığı,

- Fonksiyon ve döngü gibi kavramların daha rahat bir biçimde anlaşılmasını sağladığı,
- Öğrencilerin yaratıcılığını artırdığı,
- Dersteki başarısızlık oranını düşürdüğü,
- Derse katılımı artırdığı gözlenmiştir.

Yapılan çalışma sonuçları ve avantajları da göz önünde bulundurulduğunda, Scratch programlamayı öğrenmeye yeni adım atanlar için oldukça etkili ve önemli bir görsel programlama ortamı olarak karşımıza çıkmaktadır. Ayrıca Scratch, programlamanın zor ve soyut yanlarını, kolay, somut ve eğlenceli hale getirerek özellikle yeni başlayanlara programlamayı sevdirmekte ve programlama mantığının zihinlere yer etmesi noktasında önemli bir role sahip olmaktadır.

2.5.1.2 Alice

Son yıllarda kullanılan görsel programlama otamlarından birisi de Alice yazılımıdır. Alice, Scratch ile benzer özelliklere sahiptir. Scratch' ta olduğu gibi, kullanıcılar program söz dizimi ile uğraşmamakta, hazır kod bloklarını sürükle-bırak yöntemini kullanarak programlarını oluşturabilmektedir. Alice'in Scratch'tan farkı nesne tabanlı programlamayı öğrenmede yardımcı olmasıdır. Şekil 2.3'te görüldüğü üzere Alice 3 boyutlu bir sahneye sahiptir. Kullanıcılar karakterleri sahnenin istedikleri bir yerine sürükleyerek programlarına başlamaktadırlar. Daha sonra her karakter için detay panelinde bir takım kullanılabilecek kod blokları çıkmaktadır. Kullanıcılar bu blokları kod editörü paneline sürükleyip bırakarak programlarını hazırlayabilmektedir.



Şekil 2.3: Alice arayüzü

Programlamaya yeni başlayan öğrencilerin en fazla zorlandıkları alanlardan birisi program mantığını ve programın işleyişini yeteri kadar anlayamamalarıdır. Dann, Cooper ve Pausch (2000) bu sorunla ilgili olarak, kullanıcıların yazdıkları kod ile programın yaptığı işi ilişkilendirebilmesinin programlama sürecinde oldukça büyük bir öneme sahip olduğunu belirtmişlerdir. Bu noktada Alice yazılımı kullanıcılara çeşitli görsel olanaklar sağlayarak komut-çıktı arasındaki boşluğu etkili bir biçimde doldurmalarına fırsat tanımaktadır (Herbert, 2010). Alice'in programlamayı öğrenmeye olan katkılarını araştırmacılar şu şekilde sıralamaktadır (Brown, 2008; Cooper, Dann ve Pausch, 2000b; Sykes, 2007) :

- Söz dizimi ile ilgili hataları ortadan kaldırıp, kullanıcıların problem çözme yetenekleri üzerine yoğunlaşması,
- Görsel imkanlar sayesinde anlık geri bildirimlerle kullanıcılara program işleyişini kavrama noktasında destek sağlaması,
- Eğlenceli bir ortam sunması,
- Öğrenci güdülenmesini artıracak etkileşimli bir ara yüz sunması,
- Nesne tabanlı programlamanın görsel olarak kullanıcı zihninde temellerini atması.

Bishop-Clark, Courte, Evans ve Howard (2007) 2,5 hafta boyunca Alice yazılımını kullanan 154 öğrenciyle bir araştırma yapmıştır. Çalışmada nicel ve nitel verilerden yararlanan araştırmacılar öğrencilere Alice yazılımını kullanmadan önce ve kullandıktan sonra bir anket uygulamıştır. Öğrencilerden ayrıca Alice yazılımını değerlendiren bir yansıtma yazmaları istenmiştir. Elde edilen sonuçlara göre, Alice'in öğrencilerin programlamadaki özgüvenlerine ve temel kavramları anlamalarına anlamlı bir biçimde katkı sağladığı görülmüştür. Buna benzer bir sonuç başka bir çalışmada da elde edilmiştir (Courte, Howard ve Bishop-Clark, 2006). Araştırmacılar 100 öğrenciye Alice kullanım öncesi ve sonrasına ilişkin bir anket uygulamıştır. Çalışma sonunda alınan sonuçlar şu şekildedir. *"Genellikle programlama konusunda kendimi güvende hissederim"* ifadesi %19'dan %48'e çıkmıştır. "*Programlama dersinde özgüvenim olduğuna inanıyorum"* ifadesi %17'den % 29'a çıkmıştır. *"Programlamada iyi değilimdir"* şeklindeki bir ifade %26'dan % 16'a düşmüştür. *"Program yazmayı severim"* ifadesi %10'dan %27'e yükselmiştir. *"Programlama eğlenceli ve faydalıdır"* ifadesi ise %11'den %37'e çıkmıştır. Görüldüğü üzere, Alice öğrencilerin programlamaya karşı olumlu tutum sergileme noktasında önemli bir görev üstlenmektedir.

Sykes (2007) Alice'in programlama başarısına olan etkisini inceleyen yarı deneysel bir çalışma yapmıştır. Çalışmasında 2 kontrol grubu ve 1 deney grubu bulunmaktadır. Kontrol grupları 34, deney grubu ise 72 öğrenciden oluşmaktadır. Bütün gruplardaki öğrenciler ilk defa programlama dersi alan öğrencilerdir. Ders C programlama dili üzerinden gerçekleştirilmiştir. Kontrol grupları Alice programını kullanmazken, deney grubundaki öğrenciler haftada yarım saat Alice üzerinde çalışmıştır. Çalışma sonunda Alice yazılımını kullanan deney grubunun kontrol gruplarına göre anlamlı bir biçimde yüksek puan aldıkları görülmüştür *(p =,044).*

Moskal, Lurie ve Cooper (2004) Alice'in programlamaya giriş (CS1) dersi alan öğrencilerin programlamadaki kalıcılığına etkisini incelemişlerdir. Çalışmadaki kalıcılıktan kastedilen öğrencilerin (CS2)'ye devam edip etmemeleridir. Çalışmada 1 deney grubu 2 kontrol grubu bulunmaktadır. Deney grubu Alice yazılımını kullanırken, kontrol grupları kullanmamaktadır. Deney grubu 25, kontrol grupları ise sırasıyla 30 ve 52 öğrenciden oluşmaktadır. Deney grubundaki ve 1. kontrol grubundaki (C1) öğrenciler risk altındaki öğrencilerdir. 2. kontrol grubu C2 grubu ise risk altında olmayan öğrencilerdir. Risk altındaki öğrenciler çalışmada çok düşük

seviyede programlama ve matematik altyapısına sahip olan öğrenciler olarak tanımlanmıştır. Araştırma sonunda, Alice yazılımını kullanan risk altındaki deney grubu öğrencilerinin kalıcılık oranı %88 iken, C1 grubunda bu oran % 47, C2 grubunda ise % 75 olarak karşımıza çıkmaktadır.

Üniversite öğrencilerinin yanı sıra, Alice'in lise öğrencileri üzerindeki etkisini belirlemeye çalışan araştırmalarda mevcuttur. Wang ve diğ. (2009) yaptıkları yarı deneysel çalışmada 166 10.sınıf öğrenciler üzerinde Alice'in programlama başarısına etkisini araştırmışlardır. Rastgele olarak yapılan seçimde 2 sınıf Alice kullanırken, diğer 2 sınıf ise derslerini C++ üzerinden işlemektedirler. Araştırma 8 hafta sürmüştür. Çalışma sonunda Alice kullanan öğrencilerin anlamlı bir biçimde *(p=.02),* kullanmayan öğrencilere göre daha yüksek puan aldıkları görülmüştür.

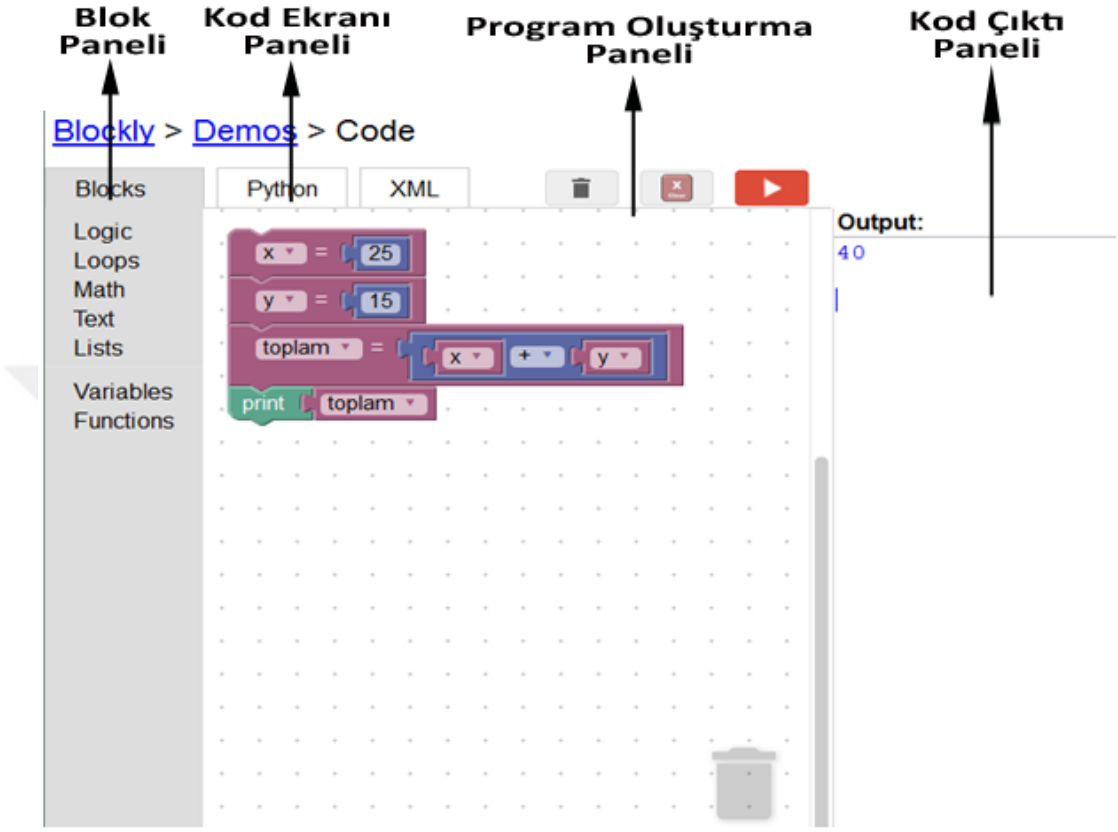
Yapılan çalışmalara bakıldığında, Alice'in özellikle programlamaya yeni adım atan öğrencilerin programlama başarısını ve kalıcılığını artırdığı görülmektedir. Ayrıca Alice'in öğrencilerin programlamadan keyif almalarını, özgüvenlerini kazanmalarını, güdülenmelerini artırmalarını ve programlamaya daha pozitif bir biçimde yaklaşmalarını sağladığı da dikkati çekmektedir. Bütün bu olumlu gelişmelerde Alice'in görsel olarak sunduğu olanakların önemli rol oynadığını söyleyebiliriz.

2.5.1.3 Blockly

Son yıllarda çıkan bir başka görsel programlama ortamı da Blockly'dir (Blockly, 2015). Blockly Google tarafından geliştirilen, Scratch ve Alice'te olduğu gibi kullanıcıyı söz diziminin zorluklarından kurtarıp bloklar ile program oluşturmaya yarayan bir ortamdır. Blockly açık kaynak kodlu bir yazılımdır. Dolayısıyla kullanıcılar Blockly üzerinde istedikleri değişiklikleri yapabilmekte ve yeni özellikler ekleyebilmektedir. Bu çalışmada Blockly üzerinde araştırmacılar tarafından çeşitli değişiklikler yapılmış ve çalışmanın amacına daha uygun bir hale getirilmiştir. Tez çalışmasında yer alan Blockly ile ilgili görsellerin büyük bir kısmı araştırmacı tarafından yapılan değişiklikler sonucu ortaya çıkan görsellerdir.

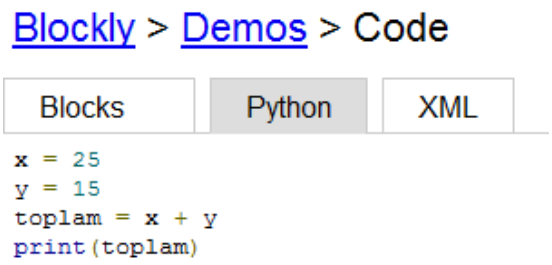
Araştırmacı tarafından düzenlenen ve bu çalışmada kullanılan Blockly yazılımı blok paneli, program oluşturma paneli, kod ekranı paneli ve çıktı panelinden oluşmaktadır. Kullanıcılar blok panelinden, mantıksal ve aritmetik ifadeler, döngüler, listeler ve işlevler gibi temel programlama kavramları ile ilgili blokları sahneye

sürükleyip bırakarak programlarını oluşturabilmektedir. Programın Python dilindeki söz dizimsel karşılıkları kod ekranı panelinde; programın çıktıları ise sağdaki *Output* panelinde görüntülenmektedir. Şekil 2.4'te Blockly ara yüzü ve iki sayının toplamını hesaplayan örnek bir program görülmektedir.



Şekil 2.4: Blockly arayüzü

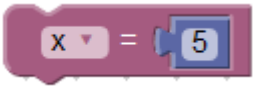
Blockly'nin Scratch ve Alice'ten farkı ve en önemli özelliklerinden birisi tasarlanan kod bloğunun JavaScript, Python, Dart ve XML dillerinde söz dizimsel olarak kodlarını vermesidir. Kullanıcılar görsel bir biçimde bloklar sayesinde program hazırlamakla kalmayıp, hazırladıkları programın sözdizimine de ulaşabilmektedir.Bu sayede kullanıcılar her bir bloğun sözdizimsel olarak karşılıklarını görebilmekte ve böylece metin tabanlı programlama dillerine daha rahat bir geçiş yapabilmektedirler. Şekil 2.5'te üstteki bloklarla hazırlanmış programın Python dilinde karşılığı görülmektedir.

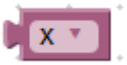


Şekil 2.5: Blockly kod ekranı

Önceden de bahsedildiği üzere kullanıcılar Blockly ile birçok programlama kavramını öğrenebilme ve bunları kullanarak program oluşturma fırsatına sahip olmaktadır. Bu noktada Blockly'nin Scratch ve Alice'e göre daha gerçekçi ve programlamayı daha ön planda tutan bir yazılım olduğu düşünülmektedir Mahadevan, Freeman ve Magerko, 2016). Bunun en önemli sebeplerinden birisi Scratch ve Alice'teki gibi sahnede yer alan karakter görünüşünü değiştirme, sahneye resim ve ses ekleme gibi programlamayı arka planda bırakacak özelliklerinin Blockly'de kullanılmasının gerekmemesidir (Meerbaum-Salant, Armoni ve Ben-Ari, 2013). Blockly'de kullanıcılar sadece programlama ile ilgili kavramlar üzerinde yoğunlaşmaktadır.

Programlama ile ilgili kavramların Blockly'de nasıl öğrenildiği aşağıdaki şekillerde bahsedilmiştir. Örneğin kullanıcılar Şekil 2.6a'da görüldüğü üzere değişken tanımlayabilmektedir. Aşağıda görüldüğü üzere ilk blokta x değişkenine 5 değeri atanmıştır. X değişkeninin programın herhangi bir yerinde kullanılması istendiğinde ise Şekil 2.6b'deki blok görev almaktadır.





Şekil 2.6a: Değişken tanımlama

Şekil 2.6b: Değişken çağırma

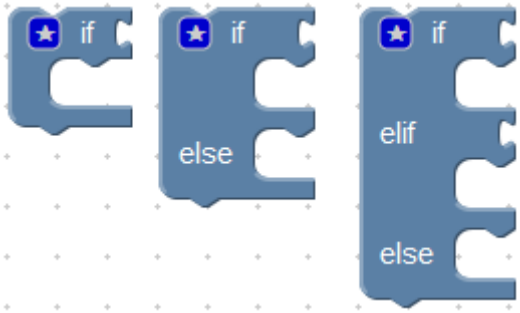
Blockly'de ayrıca programlamadaki döngü mantığını da kavrayabilmek mümkündür. Blockly *while* ve *for* döngüsünü desteklemektedir. Şekil 2.7a'da *while* döngüsüne ait blok yer almaktadır. Bu blokta kullanıcı kontrol etmek istediği durumu bloktaki ilk çentikte ve bu kontrolden sonra hangi işlemlerin yapılacağını da ikinci çentikte belirtmek durumundadır. Benzer şekilde *for* döngüsünde ise kullanıcı ne üzerinde

iterasyon (arttırma) yapılacağını belirledikten sonra döngüde hangi işlemlerin gerçekleştirileceğini belirleyebilmektedir.

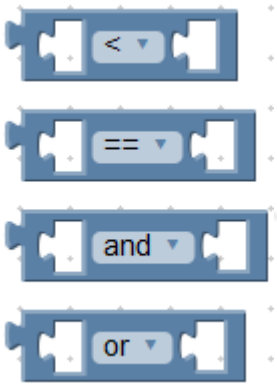
Şekil 2.7a: while döngüsü

Şekil 2.7b: for döngüsü

Blockly'de karşılaştırma ve mantıksal ifadelere de yer verilmiştir. Şekil 2.8a'da *if, if-else* ve *if-elif-else* karşılaştırma yapıları görülmektedir. Kullanıcılar burada karşılaştırma durumunu ve yapılacak işlemleri ilgili boşluklarda tanımlayabilmektedirler. Şekil 2.8b'de ise karşılaştırma durumlarında kullanılan *büyüktür, küçüktür, eşittir, ve, veya* gibi mantıksal ifadeler yer almaktadır.



Şekil 2.8a: Karşılaştırma ifadeleri



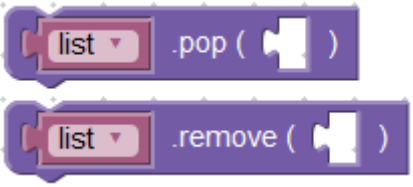
Şekil 2.8b: Mantıksal ifadeler

Bir diğer programlama kavramı olan listeler de Blockly tarafından desteklemektedir. Kullanıcılar liste bloklarıyla liste oluşturma, listeden eleman çağırma, listeden eleman silme, listedeki elemanın indeksini bulma gibi çeşitli işlemleri yapabilmektedir.

Şekil 2.9a: Liste oluşturma

Şekil 2.9b: Eleman konumunu bulma

Şekil 2.9c: Listeden eleman çağırma

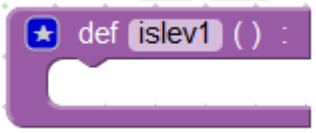


Şekil 2.9d: Listeden eleman silme

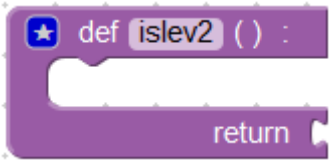
Şekil 2.9e: Listeye eleman ekleme

Şekil 2.9f: Liste dilimi alma

Kullanıcılar Blockly'de işlev kavramını da tanıma fırsatını bulabilmektedir. Blockly'de geriye bir değer döndürmeyen ve döndüren olmak üzere iki tip işlev bloğu yer almaktadır.



Şekil 2.10a: Değer döndürmeyen işlev bloğu



Şekil 2.10b: Değer döndüren işlev bloğu

Yukarıdaki şekillerden de görüldüğü gibi Blockly birçok programlama kavramı öğretimini desteklemektedir. Temel olarak her bloğun birbirleriyle bağlantılarının kurulmasıyla kullanıcılar programlarını oluşturmaktadır. Blockly'de birbirleriyle anlamsız bir biçimde blok bağlantısı mümkün değildir. Bundan dolayı hangi tip kavramların birbirleriyle ilişkilendirilebileceği de kullanıcılara daha rahat bir şekilde anlatılabilmekte ve bu sayede daha önce programlama tecrübesi yaşamayan bireylerde Blockly'den faydalanabilmektedir.

2.6 Programlama Öğreniminde İlk Dilin Etkisi

Günümüzde programlama öğretiminde Java, C, Python gibi farklı programlama dilleri tercih edilmektedir. Bununla birlikte programlama öğretimine hangi dil ile başlanması gerektiği konusunda tam olarak bir görüş birliği sağlanamasa da (Goosen, Mentz ve Nieuwoudt, 2007), özellikle programlamayı yeni öğrenecekler için uygun

bir dilin seçilmesi büyük öneme sahiptir (Mehic ve Hasan, 2001; Vujoševic-Janicic ve Tošic, 2008). Araştırmacılara göre programlamada kullanılacak ilk dilin aşağıdaki özelliklere sahip olması gerekmektedir (Böszörményi, 1998; Grandell, Peltomäki ve Salakoski, 2006; Miller ve Ranum, 2006):

- Temel programlama kavramlarını öğretmeli,
- Basit ve sade bir sözdizimine sahip olmalı,
- Problem çözme becerisini geliştirmede yardımcı olmalı,
- Programlamayı ilgi çekici hale getirmeli.

Yukarıdaki özellikler göz önünde bulundurulduğunda, çeşitli araştırmacılar Python'un programlama öğretiminde ilk dil olarak kullanılmasının daha uygun olacağından bahsetmişlerdir (Agarwal, Agarwal ve Celebi, 2005; Fangohr, 2004; Oldham, 2005; Sanders ve Langford, 2008). Bu konu ile ilgili olarak, Python'u diğer programlama dilleri ile kıyaslayan birçok çalışma bulunmaktadır. Grandell, Peltomäki ve Salakoski (2006) yaptıkları çalışmada daha önce farklı bir programlama dili tecrübesi olan öğrencilerden, o dili yeni öğrendikleri Python dili ile kıyaslamaları istenmiş ve Python'un öğrenilmesi daha kolay ve eğlenceli olduğu sonucuna varılmıştır. Fangohr (2004) ise C, MATLAB ve Python dillerini karşılaştıran bir çalışma yapmış ve Python'un diğerlerine göre daha sade bir sözdizimine sahip olduğu ve problem çözmeyi daha fazla desteklediğini belirtmiştir. Python ile Java dilini kıyaslayan bir çalışmada ise Java dilini kullanan öğrencilerin Python dilini kullanan öğrencilere göre daha çok sözdizimsel ve mantıksal hatalar yaptıkları tespit edilmiştir (Grandell, Peltomäki ve Salakoski, 2005). Patterson-McNeill (2006) ise yaptığı çalışmada kendisinin öğreticiliğini yaptığı programlamaya giriş dersinde daha önceden kullandığı C++ dilinden vazgeçip Python diline geçiş yapmış ve sonuçların önceye nazaran daha olumlu olduğunu ve öğrencilerin programlamayı daha kolay öğrendiğini belirtmiştir.

Bahsedilen çalışmalarda Python'un programlama öğretiminde ilk dil olarak kullanılmasının uygun olacağı görülebilmektedir. Bundan dolayı bu çalışmada kullanılan Blockly yazılımının Python ile tam bir şekilde uyumlu hale gelebilmesi için bir takım değişiklikler ve eklemeler yapılmıştır. Bunların tamamı EK-4'te verilmiştir.

2.7 Bölüm Özeti

Yukarıda sonuçları açıklanan çalışmalar incelendiğinde, GPO'nun öğrencilerin programlama öğrenmelerine ve programlamaya karşı tutumlarına olumlu bir biçimde katkı sağladıkları görülebilmektedir. Bununla birlikte kullanılan görsel teknolojinin etkililiğini daha sağlıklı bir biçimde ortaya çıkarmak adına bir grubun sadece görsel teknolojiyi kullandığı ve diğer grubun geleneksel yöntemle derslerine devam ettiği deneysel bir araştırma modelini ele alan çalışmalar neredeyse bulunmamaktadır (Wang ve diğ., 2009). Bu şekilde yürütülen araştırmalarda ise görsel teknolojinin bir yardımcı araç olarak kullanıldığı ve derslerin sadece küçük bir kısmında bu teknolojilerden yararlanıldığı görülebilmektedir (Ben-Bassat Levy, Ben-Ari ve Uronen, 2003; Sykes, 2007). Diğer bir deyişle bu tür teknolojilerin yardımcı bir araç olarak kullanılmadığı ve programlama derslerinin sadece bu tür teknolojiler üzerinden işlendiği araştırmalar yok denecek kadar azdır. Ancak bu çalışmada belirlenen deney grubu hem ders hem de laboratuar saatlerinde sadece Blockly yazılımını kullanmakta; kontrol grubu ise herhangi bir görsel teknoloji kullanmadan geleneksel bir şekilde derslerini işlemektedirler. Bu şekilde ele alınan bir araştırma modelinin, Blockly'nin öğrencilerin programlama öğrenmelerine olan katkısını daha gerçekçi ve güvenilir bir biçimde sunacağı düşünülmektedir.

Bunlara ek olarak programlamada görsel programlama ortamlarının etkinliğini inceleyen çalışmalar çoğunlukla Scratch ve Alice yazılımını kullanmaktadırlar. Bu yazılımlara bakıldığında programlama mantığının temellerini atmalarının yanı sıra programlama ile ilgisi olamayan bazı özellikleri *(karakter görünüşünü değiştirme, sahneye resim ya da müzik ekleme vb.)* barındırdığı görülebilmektedir. Ancak Blockly yazılımı hem yapısı itibariyle hem de oluşturulan programların gerçek söz dizimsel karşılıklarını da vermesi ile sadece programlamayı öğretmeyi hedef alan, ve programlama dışı özelliklere sahip olmayan bir görsel programlama ortamı olarak karşımıza çıkmaktadır. Bu özellikleri ile Scratch ve Alice'ten ayrıldığı ve öğrencilere daha gerçekçi bir programlama ortamı sunduğu düşünülmektedir. Ayrıca Blockly yazılımının programlama öğretimindeki etkisinin incelenmediği de görülmüştür. Bu sebeple bu tarz özelliklere sahip bir görsel programlama aracının öğrencilerin programlamayı öğrenmelerine ve programlamaya karşı olan tutumlarına olan etkilerini incelemek ve bu alanda literatüre bir katkı sağlamak hedeflenmiştir.

BÖLÜM 3
ARAŞTIRMA PROBLEMİ VE HİPOTEZLER

Bu kısımda çalışmanın ana problemi ve hipotezler yer almaktadır.

3.1 Ana Problem ve Hipotezler

Bu çalışmada ana problem ve ana problemi test etmek için 2 hipotez bulunmaktadır.

3.1.1 Ana Problem

Görsel programlama öğretimi (GPÖ) ve geleneksel programlama öğretiminin (GEPÖ) öğrencilerin programlamayı öğrenmesine ve programlamaya karşı tutumlarına etkisi arasında anlamlı bir fark var mıdır?

3.1.2 Hipotezler

Bu çalışmadaki hipotezler aşağıda verilmiştir. Hipotezler sıfır hipotezi olarak 0.05 anlamlılık düzeyinde tanımlanmıştır.

$H_0$1: GPÖ ve GEPÖ'nün öğrencilerin programlamayı öğrenmesi üzerindeki etkilerinin arasında anlamlı bir fark yoktur.

$H_0$2: GPÖ ve GEPÖ ile öğrenim gören öğrencilerin programlamaya karşı tutumları arasında anlamlı bir fark var yoktur.

BÖLÜM 4
YÖNTEM

Bu bölümde çalışmanın deseni, çalışma grubu, değişkenler, veri toplama araçları, uygulama süreci, veri analiz yöntemi, varsayımlar ve sınırlılıklar yer almaktadır.

4.1 Yarı Deneysel Desen

Görsel teknoloji kullanımının öğrencilerin programlama öğrenmelerine ve programlamaya karşı tutumlarına olan etkisini inceleyen bu çalışmada nicel araştırma yöntemlerinden yararlanılmıştır. Araştırmada ön test-son test, kontrol gruplu yarı deneysel bir desen benimsenmiştir. Deneysel araştırmalar, bağımsız değişkenin araştırmacı tarafından manipüle edildiği ve deneklerin iki veya daha fazla koşulda bağımlı değişkene ilişkin ölçümlerinin karşılaştırıldığı araştırma desenidir (Büyüköztürk, Çakmak, Akgün, Karadeniz ve Demirel, 2008). Yarı deneysel desen ise seçkisiz atamanın mümkün olmadığı bir deneysel araştırma desenidir (Büyüköztürk ve diğ., 2008). Araştırmanın deseni Tablo 4.1'de görülebilmektedir.

Tablo 4.1: Araştırma deseni

Grup	Ön Test	Uygulanan Yöntem	Son Test
Deney grubu	Programlama Yetenek Testi (PYT)	Görsel Programlama Öğretimi (GPÖ)	Programlama Başarı Testi (PBT), Bilgisayar Programlamaya Karşı Tutum Ölçeği (BPKTÖ)
Kontrol grubu	Programlama Yetenek Testi (PYT)	Geleneksel Programlama Öğretimi (GEPÖ)	Programlama Başarı Testi (PBT), Bilgisayar Programlamaya Karşı Tutum Ölçeği (BPKTÖ)

Deney ve kontrol grubuna öğretim uygulamalarından önce ve sonra yukarıda belirtilen testler uygulanmıştır. Deney grubunda programlama dersi bir görsel

programlama öğretim aracı olan Blockly yazılımı üzerinden işlenirken, kontrol grubunda dersler geleneksel programlama öğretimi kapsamında işlenmiştir.

4.2 Çalışma Grubu

Bu çalışmanın gereklerinden birisi katılımcıların daha önceden programlama tecrübesi yaşamamalarıdır. Bundan dolayı daha önce programlama öğretimi görmeyen Ondokuz Mayıs Üniversitesi Bilgisayar ve Öğretim Teknolojileri Eğitimi Bölümü ikinci sınıf öğrencileri çalışmanın katılımcılarını oluşturmaktadır. Çalışma Programlama Dilleri I dersi kapsamında gerçekleştirilmiştir. Programlama I dersinde öğrenciler numaralarının son rakamı tek olanlar A ve çift olanlar B grubu şeklinde oluşturulmuştur. Gruplar haftanın farklı günlerinde eğitim görmektedirler. Deney ve kontrol grupları seçilirken rastgele seçilmiş olup; A grubundaki öğrenciler deney grubu, B grubundaki öğrenciler ise kontrol grubu olarak eğitim görmüşlerdir. Ara sınava kadar deney grubu ders ve laboratuar saatlerinde Blockly programıyla programlama öğretimi almışken, kontrol grubu ise ders ve laboratuar saatlerinde geleneksel öğretim tarzı ile Python programlama dili kullanılarak öğrenim görmüşlerdir. Öğretimi bu tezin danışmanı olan öğretim üyesi gerçekleştirmiştir. Her iki grupta anlatım tarzı ve verilen örnekler aynıdır. Gruplar arasındaki tek fark deney ve kontrol grubunda kullanılan öğrenme ortamlarıdır (Blockly ve Python). Çalışma grubuna daha önceden programlama bilenler, derslere devam etmeyenler ve bazı testlere katılmayan öğrenciler dahil edilmemiştir. Bu kişiler çıkartıldıktan sonra oluşan çalışma grubu 42 öğrenci olarak belirlenmiştir. Grupların dağılımı Tablo 4.2'de verilmiştir.

Tablo 4.2: Çalışma grubu

Cinsiyet	Deney grubu	Kontrol grubu	Toplam
Kız	13	8	21
Erkek	9	12	21
Toplam	22	20	42

Ara sınavdan sonra her iki grup da geleneksel öğretimle programlamayı öğrenmeye devam etmişlerdir. Ara sınava kadar işlenen konular ile ilgili kazanımlar aşağıdaki gibidir. Konu başlıkları ve içerikleri aşağıdaki gibi olup Başer (2013c)'den alınmıştır.

1. Bölüm: Değişkenler

Bu bölüm sonunda öğrenciler;

- Değişkenin tanımını yapar.
- Değerler ve bunların veri tiplerini belirtir.
- İşleç ve ifade terimlerinin tanımını yapar.
- Değişken tanımlar.
- Python'daki anahtar kelimeleri fark eder.
- Matematiksel işlemleri yapar.
- Tipler arası dönüşüm yapar.

2. Bölüm: İşlevler

Bu bölüm sonunda öğrenciler;

- İşlev tanımlar.
- İşlevi çağırır.
- İşlevden geri dönme *(return)* ifadesini kullanır.
- İşlevlere veri gönderir.
- İşlevlerden veri döndürür.
- Açıklama satırlarını kullanır.
- Python kodlarını program haline getirir.
- İşlevlerin ön tanımlı parametrelerini kullanır.

3. Bölüm: Karşılaştırmalar ve while Döngüsü

Bu bölüm sonunda öğrenciler;

- Karşılaştırmanın ne olduğunu ifade eder.
- Karşılaştırma işleçlerinin ne olduğunu ifade eder.
- Mantıksal işleçlerin ne olduğunu ifade eder.
- *if, if-else, if-elif-else* deyimlerini kullanır.
- Zincirleme karşılaştırmalar yapar.
- Tekrarlamanın nasıl yapılacağını ifade eder.
- Döngüyü tanımlar.
- while döngüsünden koşullu çıkma işlemi yapar.

4. Bölüm: Listeler ve İterasyon

Bu bölüm sonunda öğrenciler;

- Python'da nesne tanımını yapar.
- Nesnelerin özelliklerini fark eder.
- Liste tanımlar.
- Listeye eleman ekler.
- Listeden eleman çıkarır.
- Listedeki elemanın yerini belirler.
- Listedeki elemanları sıralar.
- Listede bir elemanın olup olmadığını denetler.
- Liste elemanları üzerinde iterasyon (*for* deyimi) yapar.
- Nesneleri takma isimleriyle çağırır.
- Listeden dilim alır.
- İç içe liste (matris) kullanımını ifade eder.

4.3 Değişkenler

Bu çalışmada bir bağımsız değişken iki adet bağımlı değişken bulunmaktadır.

4.3.1 Bağımsız Değişken

Bu çalışmadaki bağımsız değişken deney ve kontrol gruplarına uygulanan öğretim yöntemidir (GPÖ ve GEPÖ). Öğrencilerin Programlama Yetenek Testi'nden almış oldukları puanlar ise ortak değişken *(covariate)* olarak değerlendirilmiştir.

4.3.2 Bağımlı Değişkenler

Bu çalışmadaki bağımlı değişkenler öğrencilerin Programlama Başarı Testi ve Programlamaya Karşı Tutum Ölçeği'nden almış oldukları son test puanlarıdır.

4.4 Veri Toplama Araçları

Çalışmanın başında öğrencilere ön test olarak Programlama Yetenek Testi uygulanmıştır (Computer Programming Aptitude Test, 2015). Son test olarak ise Programlama Başarı Testi ve Programlamaya Karşı Tutum Ölçeği (Başer, 2013a) kullanılmıştır.

4.4.1 Programlama Yetenek Testi (PYT)

Öğrenciler daha önce programlama bilmediklerinden deney ve kontrol grubu öğrencilerinin programlama yeteneklerinin çalışmanın başında eşit olup olmadığını belirlemek için Kent Üniversitesi Kariyer Danışma Birimi tarafından geliştirilen PYT uygulanmıştır. Kent Üniversitesi'nde programlama dersini almak isteyen öğrencilere öncelikle bu test uygulanmaktadır. Bu testten yeterli puanı alabilen öğrenciler programlama dersine kayıt yaptırma hakkına sahip olabilmektedir. Programlama yetenek testi öğrencilerin programlamayı öğrenip öğrenemeyeceklerini önceden tahmin edebilmek için kullanılmaktadır (Tukiainen ve Mönkkönen, 2002). Bu sebeple, bu çalışmada da öğrencilerin programlamayı öğrenme kapasiteleri uygulanan öğretim uygulamasının etkililiğini belirleme noktasında önem arz ettiğinden çalışmada yer alan öğrencilere Programlama Yetenek Testi ön test olarak uygulanmıştır.

Programlama Yetenek Testi'nde 26 soru yer almaktadır. Bu sorular mantıksal düşünme, sayısal düşünme, desen tanıma ve detaylara dikkati ölçen ve herhangi bir programlama ön bilgisi gerektirmeyen sorulardır (Computer Programming Aptitude Test, 2015). Testin Türkçeye çevrilmesi ve uyarlanması araştırmacılar tarafından yapılmıştır. Testin güvenirlik düzeyini belirlemek için, test daha önce programlama dersi almamış 42 öğrenciye uygulanmıştır. Yapılan güvenirlik analizi sonucunda testin Cronbach-α güvenirlik katsayısı 0,647 olarak bulunmuştur. Testin güvenirliğini geliştirmek için madde-toplam korelasyonu 0,2'den düşük olan 4 maddenin testten çıkarılmasına karar verilmiştir (Everitt ve Skrondal, 2002, s 225). Bu maddeler çıkarıldıktan sonra testin Cronbach-α güvenirlik katsayısı 0,707 olarak hesaplanmıştır. Bu değere göre testin iki grubun başarılarını karşılaştırmak için güvenilir olduğu söylenebilir (Hair ve diğ., 2010, s 123). Testten alınabilecek en yüksek puan 100, en düşük puan ise 0'dır.

4.4.2 Bilgisayar Programlamaya Karşı Tutum Ölçeği (BPKTÖ)

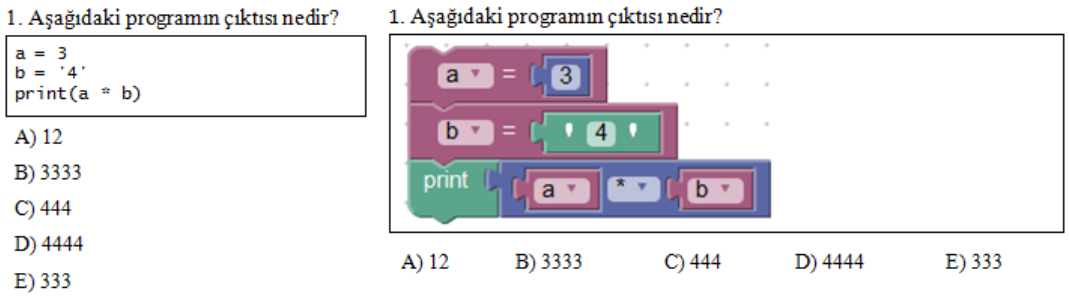
Deney ve kontrol grubuna uygulanacak iki farklı öğretim biçiminin öğrencilerin programlamaya karşı tutumlarını nasıl etkiledikleri de bu çalışmada cevaplanması gerektiğine inanılan bir durumdur. Bundan dolayı bu soruya cevap bulabilmek amacıyla Başer'in (2013a) geliştirmiş olduğu BPKTÖ son test olarak uygulanmıştır.

Öğrenciler daha önce programlama dersi almadıklarından, başlangıçta programlamaya karşı tutumlarını öğrenme şansı yoktur. Öğrencilerin Bilgisayar Programlamaya Karşı Tutumları bu dersin içerisinde oluşacağı düşünülmektedir.

Ölçekte 38 ifade bulunmaktadır. Her bir ifade katılımcılar tarafından *"Kesinlikle Katılmıyorum", "Katılmıyorum", "Karasızım", "Katılıyorum"* ve *"Kesinlikle Katılıyorum"* şeklinde değerlendirilecektir. Yapılan analizlerde Başer (2013a) ölçek maddelerinin "*Programlamada kendine güven ve güdülenme", "Programlamanın faydası", "Programlamada başarıya karşı tutum" ve "Programlamada başarının sosyal algısı"* şeklinde 4 boyut altında toplandıkları sonucuna varmıştır. Alt boyutların tamamının Cronbach-α güvenirlik katsayıları sırasıyla 0.944, 0.920, 0.926, 0.618 olup ölçeğin tamamının Cronbach-α güvenirlik katsayısı ise 0,953 olup güvenilir bir ölçek olarak nitelendirilebilmektedir (Hair ve diğ., 2010, s 123).

4.4.3 Programlama Başarı Testi (PBT)

Geliştirilen öğretim uygulamasının öğrencilerin programlama öğrenmelerine olan etkisini tespit etmek amacıyla deney ve kontrol grubundaki öğrencilere araştırma sonunda son test olarak araştırmacılar tarafından geliştirilen PBT uygulanmıştır. Testte 25 soru yer almaktadır. Testteki sorular deney ve kontrol grubu için aynıdır. Ancak deney grubundaki öğrenciler soruların Blockly'de hazırlanmış şeklini çözerken, kontrol grubundaki öğrenciler soruların Python'da yazılmış biçimini çözmüşlerdir. Şekil 4.1'de bir sorunun sırasıyla kontrol ve deney grupları için hazırlanmış şekli görülebilmektedir.



Şekil 4.1: Programlama başarı testi örnek soru

Yapılan güvenirlik analizi sonucunda Programlama Başarı Testi'nin Cronbach-α güvenirlik katsayısı 0,774 olarak bulunmuştur. Bu değer testin grup karşılaştırmasında güvenle kullanılabileceğini göstermektedir (Hair ve diğ., 2010, s 123). Test maddelerinin ayırt edicilikleri Tablo 4.3'te verilmiştir.

Tablo 4.3: PBT Madde Ayırt Edicilik Katsayıları

Soru	Madde ayırt edicilik katsayısı	Soru	Madde ayırt edicilik katsayısı
Soru1	0,29	Soru14	0,72
Soru2	0,52	Soru15	0,59
Soru3	0,31	Soru16	0,24
Soru4	0,39	Soru17	0,50
Soru5	0,41	Soru18	0,52
Soru6	0,18	Soru19	0,58
Soru7	0,47	Soru20	0,53
Soru8	0,14	Soru21	0,72
Soru9	0,37	Soru22	0,42
Soru10	0,21	Soru23	0,54
Soru11	0,32	Soru24	0,54
Soru12	0,26	Soru25	0,58
Soru13	0,35		

Jha (2014) madde ayırt edicilik katsayısı 0.2'den az olanların çıkarılması gerektiğini belirtmiştir. Bu durumda madde ayırt edicilik katsayısı 0.2'den az olarak 6. ve 8. sorular görülmektedir. 8. soru çok kolay olup ayırt ediciliği düşük olduğu için çıkartılıp analize dahil edilmemiştir. 6. soru ise *"zincirleme if "* ifadelerinin öğrenilip öğrenilmediğini test etmek için gerekli bir sorudur. Ayrıca *"if-if"* ile *"if-elif"* yapıları arasındaki farkı test etmek için de gerekli görülüp analizden çıkartılmamıştır. Madde ayırt edicilik katsayısı düşük olup araştırmacı tarafından önemli görülen maddelerin çıkartılmayabileceği alanyazında belirtilmiştir (Miller, Linn ve Gronlund, 2008). Testten alınabilecek en yüksek puan 100, en düşük puan ise 0'dır. Testteki sorular özdeş olup, deney grubunda Blockly blokları, kontrol grubunda Python kodları vardır. Bunların aynı öğrenme çıktısını ölçüp ölçemeyeceği, Ölçme ve Değerlendirme uzmanına sorulmuştur. Kendisi her iki soru maddelerinin de aynı öğrenme çıktısını ölçtüğünü belirtmiştir.

4.5 Uygulama Süreci

Uygulama görsel programlama öğretiminin öğrencilerin bilgisayar programlamayı öğrenmesine ve programlamaya karşı tutumlarının incelenmesi amacıyla 2015/2016 eğitim öğretim yılı 1. döneminde Bilgisayar ve Öğretim Teknolojileri Eğitimi Bölümü (BÖTE) öğrencileriyle Programlama Dilleri I dersi kapsamında gerçekleştirilmiştir. Uygulamada deney ve kontrol grubu olmak üzere iki grup bulunmaktadır. Uygulama 8 hafta boyunca devam etmiştir. Her hafta derse ait 3 saat teorik anlatım kısmı, 2 saat de laboratuar etkinlikleri kısmı bulunmaktadır. Deney grubu öğrencileri bir görsel programlama öğretim aracı olan Blockly ile programlama derslerini sürdürürken, kontrol grubu ise geleneksel programlama öğretimi gören öğrencilerden oluşmaktadır. Derste anlatılan konular ve laboratuar saatlerinde öğrenciler tarafından uygulanan etkinlikler deney ve kontrol grubu için birebir aynıdır. İki grup arasındaki tek fark programlama öğretim aracıdır.

4.6 Veri Analizi

Bu araştırmada görsel teknoloji kullanımının öğrencilerin programlama öğrenmelerine ve programlamaya karşı tutumlarına olan etkisi incelenmiştir. Veri toplama araçlarında öğrencilerin vermiş olduğu cevaplar bilgisayar ortamına aktarılmış ve bu sayede daha doğru ve sağlık bir biçimde sonuçlar elde edilmiştir. Elde edilen sonuçlar SPSS 21 programı kullanılarak analiz edilmiştir.

Elde edilen veriler 0,05 anlamlılık düzeyine göre incelenmiştir. Verilerin analizinde farklı istatistik tekniklerinden yararlanılmıştır. Görsel teknoloji kullanımının öğrencilerin programlama başarısına olan etkisi incelenirken kovaryans analizi (ANCOVA) kullanılmıştır. Görsel teknoloji kullanımının öğrencilerin programlamaya karşı olan tutumlarına etkisi incelenirken ise bağımsız örneklemler t-testin ve Mann-Whitney U testinden faydalanılmıştır. Kullanılacak olan istatistik tekniğinin varsayımlarının karşılanmasına dikkat edilmiştir.

4.7 Varsayımlar ve Sınırlılıklar

Çalışmanın varsayımları ve sınırlılıkları aşağıda verilmiştir.

4.7.1 Varsayımlar

1. Çalışmada kullanılan veri toplama araçlarının hedeflenen ölçümleri geçerli güvenilir bir biçimde ölçtüğü varsayılmıştır.
2. Veri toplama araçlarının çalışmaya katılan öğrenciler tarafından içtenlikle cevaplandırıldığı varsayılmıştır.
3. Deney ve kontrol grubundaki öğrencilerin çalışma sonucunu değiştirecek bir biçimde birbirleriyle etkileşimde bulunmadıkları varsayılmıştır.
4. Kontrol altına alınamayan değişkenlerin deney ve kontrol grubundaki öğrencileri eşit bir şekilde etkilediği varsayılmıştır.

4.7.2 Sınırlılıklar

1. Çalışmanın katılımcıları Ondokuz Mayıs Üniversitesi Bilgisayar ve Öğretim Teknolojileri Bölümü'nde eğitim gören 2. sınıf düzeyindeki 42 öğrenciyle sınırlıdır.
2. Çalışmanın süresi 8 hafta ile sınırlıdır.
3. Çalışma Programlama Dilleri I dersi kapsamında yer alan "Değişkenler", "İşlevler", "Karşılaştırmalar ve while Döngüsü" ve "Listeler ve İterasyon" konuları ile sınırlıdır.

BÖLÜM 5
BULGULAR

Bu bölümde çalışma hipotezlerinin test edilmesi sonucu ulaşılan sonuçlar yer almaktadır. Hipotezler 0.05 anlamlılık düzeyinde test edilmiştir. Hipotez testlerinde kovaryans analizi (ANCOVA), t-test ve Mann-Whitney U testi kullanılmıştır. Analizler SPSS 21 istatistik yazılımı yardımıyla yapılmıştır.

5.1 Bulgular

Görsel teknoloji kullanımının öğrencilerin programlama öğrenmelerine ve programlamaya karşı tutumlarına etkisini incelemek amacıyla deney ve kontrol gruplarına ön test olarak Programlama Yetenek Testi (PYT); son test olarak da Programlama Başarı Testi (PBT) ve Bilgisayar Programlamaya Karşı Tutum Ölçeği (BPKTÖ) uygulanmıştır. Gruplara göre düzenlenmiş ön test ve son testlere ait betimsel istatistikler Tablo 5.1'de verilmiştir.

Tablo 5.1: Gruplara göre düzenlenmiş betimsel istatistikler

	PYT		**PBT**		**BPKTÖ**	
	DG	**KG**	**DG**	**KG**	**DG**	**KG**
N	22	20	22	20	22	20
$\bar{X}$	60.96	67.50	66.40	61.08	3.91	3.88
Medyan	61.37	70.46	67.39	60.87	3.89	3.77
Minimum	40.91	40.91	34.78	26.09	3.16	3.00
Maksimum	90.91	90.91	86.96	82.61	4.68	4.55
Standart Sapma	12.36	14.84	13.70	15.74	0.44	0.40
Çarpıklık	.367	-.336	-.477	-.607	.182	-.075
Basıklık	.288	-.574	-.068	-.405	-.783	-.253

Çalışmaya başlamadan önce grupların eşit düzeyde olup olmadığını test etmek için t-test uygulanmıştır. Verilerin normal dağılıp dağılmadığı Tablo 5.2'deki Shapiro-Wilk normallik testinde verilmiştir. Ön teste ait veriler analiz edildiğinde deney ve kontrol

grupları arasında Tablo 5.3'te görüldüğü üzere anlamlı bir fark bulunmadığı tespit edilmiştir. (p>.05).

Tablo 5.2: Ön teste ait Shapiro-Wilk Normallik Testi

				Shapiro-Wilk		
Grup	**N**	$\bar{X}$	**Ss**	**İstatistik**	**sd**	**p**
Deney	22	60.96	12.36	.963	22	.546
Kontrol	20	67.50	14.84	.944	20	.288

Tablo 5.3: Ön teste ait İlişkisiz Örneklemler t-Testi Sonuçları

Grup	**N**	$\bar{X}$	**Ss**	**sd**	**t**	**p**
Deney	22	60.96	12.36	40	-1.559	.127
Kontrol	20	67.50	14.84			

Hipotez 1:

Çalışmanın hipotezlerinden biri olan "GPÖ ve GEPÖ'nün öğrencilerin programlamayı öğrenmesi üzerindeki etkilerinin arasında anlamlı bir fark yoktur" hipotezinin test edilmesi için kovaryans analizi (ANCOVA) kullanılmıştır. Bu çalışmada uygulanan deneysel işlemin etkili olup olmadığına odaklanılmıştır. Bundan dolayı, uygulamanın başında öğrenciler programlama bilmemelerine rağmen Programlama Yetenek Testi sayesinde ölçülen programlama öğrenebilme kapasitelerinin programlama öğrenmeleri üzerinde etkisinin olabileceği düşünülmüş ve bu etkinin ortadan kaldırılarak daha sağlıklı bir sonuç elde edilebilmesi için ANCOVA tercih edilmiştir (Field, 2013). Tanım olarak ANCOVA araştırmacının deneysel yöntemin etkililiğini belirlemek için kullandığı ve bağımlı değişken ile ilişkisi olduğu düşünülen değişkenlerin istatistiksel olarak kontrol edildiği bir istatistik tekniğidir (Büyüköztürk, 2014; Field, 2013). Kovaryans analizinde bağımlı değişken üzerinde etkisi kontrol edilecek değişkene ortak değişken (covariate) adı verilir. Bu analiz ortak değişkenlerin etkisini ortadan kaldırarak uygulanan deneysel işlemin gerçek etkisinin ortaya çıkmasını mümkün kılar (Field, 2013). Bu çalışmada bağımlı değişken öğrencilerin PBT'den aldıkları son test puanları; bağımsız değişken

deney ve kontrol grubuna uygulanan öğretim yöntemi; ortak değişken de PYT'den alınan ön test puanlarıdır.

Kovaryans analizinin uygulanabilmesi için aşağıda belirtilen varsayımların karşılanması gerekmektedir :

- Çalışmadaki her bir grup için bağımlı değişkene ait puanların evrendeki dağılımı normaldir.
- Çalışmadaki her bir grup için bağımlı değişkene ait varyanslar eşittir.
- Gruplar içi regresyon eğimleri eşittir.
- Bağımlı ve ortak değişken arasından doğrusal bir ilişki vardır.
- Ortalama puanları karşılaştırılacak olan örneklemler ilişkisizdir (Büyüköztürk, 2014; Field, 2013).

Birinci varsayım için her iki grubun son test puanlarının normal dağılıp dağılmadığı Shapiro-Wilk testi ile incelenmiştir. Shapiro-Wilk testi örneklem sayısının 50'den az olduğu durumlarda tercih edilen bir testtir (Büyüköztürk, 2014). Normallik testine ilişkin sonuçlar Tablo 5.4'te verilmiştir. Tablo 4.3'teki sonuçlara bakıldığında deney grubu ($p>.05$) ve kontrol grubunun ($p>.05$) son test puanları normal dağılım göstermektedir. Ayrıca Tablo 5.1'deki grupların PBT'ye ait çarpıklık ve basıklık değerlerinin standart hatalarına oranı da ±1.96 aralığında olması her iki grubun son test puanlarının normal dağılım sergilediğini teyit etmektedir (Field, 2013, s 139; Hair, Black, Babin ve Anderson, 2010, s 71). Buna göre kovaryans analizi için gerekli olan bağımlı değişkene ait puanların normal dağılım varsayımının karşılandığı görülmektedir.

Tablo 5.4: Deney ve Kontrol gruplarının PBT Shapiro-Wilk Testi Sonuçları

				Shapiro-Wilk		
Grup	**N**	$\bar{X}$	**Ss**	**İstatistik**	**sd**	**p**
Deney	22	66.40	13.69	.957	22	.425
Kontrol	20	61.08	15.74	.939	20	.227

İkinci varsayım olarak, grupların son testlerine ait varyansların eşit olup olmadığı incelenmiştir. Bunun için Levene Testi kullanılmıştır ve test sonuçları aşağıdaki tabloda verilmiştir.

Tablo 5.5: Levene testi sonuçları

Test	F	sd1	sd2	p
Programlama Başarı Testi	.155	1	40	.696

Tablo 5.5'teki sonuçlara göre deney grubu ve kontrol grubunun son test puanlarına ait varyansların eşit olduğu sonucuna varılabilmektedir, $F(1, 40) = 0.155$, $p>.05$.

Kovaryans analizinin uygulanabilmesi için gruplar içi regresyon eğimlerinin de eşit olması gerekmektedir. Bunu test etmek amacıyla uygulanan ANOVA sonuçları Tablo 5.6'da verilmektedir.

Tablo 5.6: Regresyon Eğimleri Eşitliği İçin ANOVA Sonuçları

Varyansın Kaynağı	Kareler Toplamı	sd	Kareler Ortalaması	F	p
Grup	82.457	1	82.457	.570	.455
Ön Test	3039.855	1	3039.855	20.002	.000
Grup*Ön Test	8.308	1	8.308	.057	.812
Hata	5500.056	38	144.738		
Toplam	180282.446	42			

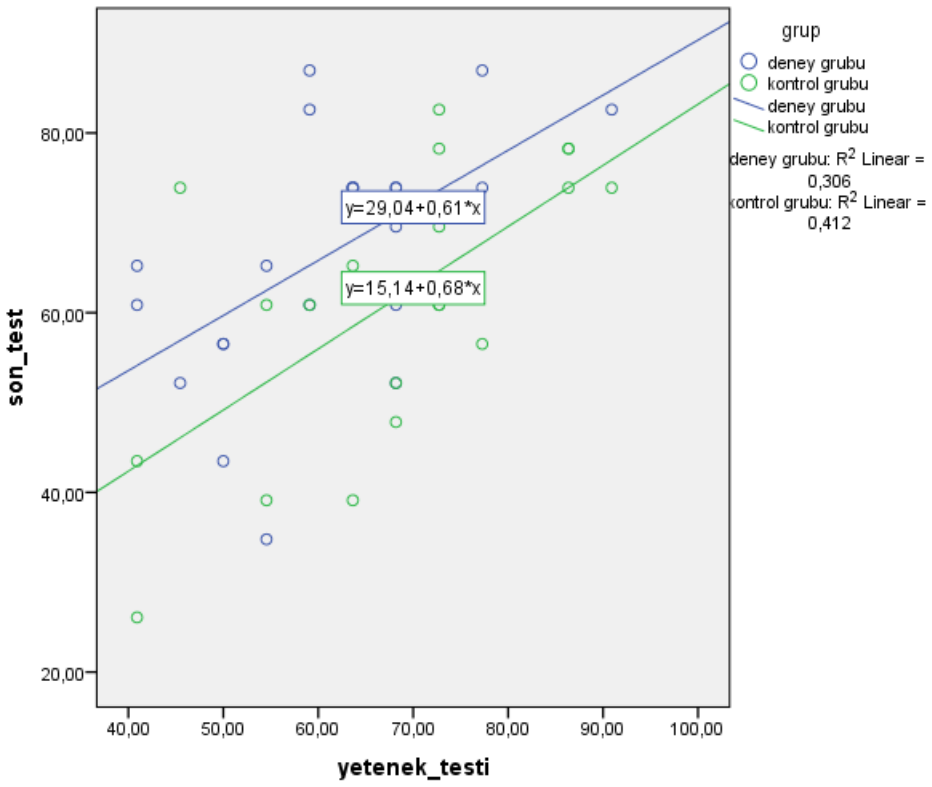
Regresyon eğimlerine ait sonuçlar incelendiğinde öğrencilerin son test programlama başarı puanları üzerinde Grup*Ön Test ortak etkisinin anlamsız olduğu $F(1, 38) = 0.057$, $p>.05$ ve dolayısıyla gruplar içi regresyon eğimlerinin eşit olduğu görülmektedir (Büyüköztürk, 2014).

Kovaryans analizinin uygulanabilmesi için ayrıca bağımlı ve ortak değişken arasında doğrusal bir ilişkinin var olması gerekmektedir. Bunu test etmek için Pearson korelasyon analizi yapılmıştır. Yapılan analiz sonucu Tablo 5.7'de verilmiştir.

Tablo 5.7: Bağımlı ve Ortak Değişken Arasındaki Doğrusal İlişki

Grup	Değişkenler	N	r	p
Deney	PBT x PYT	22	0.553	.008
Kontrol	PBT x PYT	20	0.642	.002

Tablo 5.7 incelendiğinde deney ve kontrol grubuna ait bağımlı değişken ve ortak değişken arasında istatistiksel olarak anlamlı bir doğrusal ilişkinin olduğu görülmektedir ($p<.05$). Ayrıca değişkenler arasındaki doğrusallık Şekil 5.1'deki saçılma grafiğinde de görülebilmektedir.



Şekil 5.1: Ön Test - Son Test Saçılma Grafiği

Son varsayım olarak ortalama puanları karşılaştırılacak örneklemler ilişkisiz olmalıdır. Bu çalışmadaki deney ve kontrol grupları birbirinden bağımsız iki gruptur.

Bütün varsayımların karşılandığı tespit edildikten sonra ANCOVA uygulanmıştır. Bu analize ilişkin sonuçlar Tablo 5.8'de verilmiştir.

Tablo 5.8: Son Teste ait ANCOVA sonuçları

Varyansın Kaynağı	Kareler Toplamı	sd	Kareler Ortalaması	F	p
Ön Test	3136.563	1	3136.563	22.207	.000
Grup	906.752	1	906.752	6.420	.015
Hata	5508.364	39	141.240		
Toplam	180282.446	42			

Tablo 5.8'de yer alan ANCOVA sonuçlarına göre deney ve kontrol gruplarının ön test puanları kontrol altına alındığında grupların son test puanlarının karşılaştırılması sonucunda deney grubu lehine anlamlı bir farka rastlanmıştır, $F(1, 39) = 6.420$, $p<.05$. Buna göre görsel programlama öğretimi alan öğrencilerin geleneksel yönteme göre programlama öğretimi alan öğrencilere göre istatistiksel olarak anlamlı bir biçimde daha başarılı oldukları sonucuna varılabilmektedir.

Hipotez 2:

"GPÖ ve GEPÖ ile öğrenim gören öğrencilerin programlamaya karşı tutumları arasında anlamlı bir fark var yoktur" hipotezinin test edilmesi için ise ilişkisiz (bağımsız) örneklemler t-testi kullanılmıştır. İlişkisiz örneklemler t-testi bir değişkene ilişkin oluşan grupların bir bağımlı değişkene ait ölçümlerinin karşılaştırılmasında ve gruplar arasındaki farkın istatistiksel olarak anlamlı olup olmadığını test etmek için kullanılır (Büyüköztürk, 2014; Field, 2013). Bu teste ait karşılanması gereken bir takım varsayımlar bulunmaktadır:

- Bağımlı değişkene ait ölçümler aralık ya da oran ölçeğindedir ve karşılaştırılacak iki grup ortalaması aynı değişkene aittir.
- Bağımlı değişkene ilişkin ölçümler her iki grupta da dağılımı normaldir.
- Ortalama puanları karşılaştırılacak örneklemler ilişkisizdir (Büyüköztürk, 2014; Field, 2013).

Bu hipotezin testi için her iki gruba son test olarak Bilgisayar Programlamaya Karşı Tutum Ölçeği verilmiştir. Bu test ile ilgili ölçümler aralık ölçeğindedir. Dolayısıyla ilişkisiz örneklemler t-testi için gerekli olan birinci varsayım karşılanmaktadır.

İkinci varsayım olarak, bağımlı değişkene ait ölçümlerin normal dağılıp dağılmadığını belirlemek amacıyla Shapiro-Wilk testi uygulanmıştır. Bu teste ilişkin sonuçlar Tablo 5.9'da verilmiştir.

Tablo 5.9: Bilgisayar Programlamaya Karşı Tutum Ölçeği Shapiro-Wilk Normallik Testi Sonuçları

				Shapiro-Wilk		
Grup	**N**	**$\bar{X}$**	**Ss**	**İstatistik**	**sd**	**p**
Deney	22	3.91	0.44	.961	22	.511
Kontrol	20	3.88	0.41	.957	20	.410

Tablo 5.9'daki sonuçlara bakıldığında deney grubu (p>.05) ve kontrol grubunun (p>.05) Bilgisayar Programlamaya Karşı Tutum Ölçeği son test puanları normal dağılım göstermektedir. Buna göre t-test analizi için gerekli olan bağımlı değişkene ait puanların normal dağılım varsayımının karşılandığı görülmektedir.

Üçüncü varsayım olarak ortalama puanları karşılaştırılacak örneklemler ilişkisiz olmalıdır. Bu çalışmadaki deney ve kontrol grupları birbirinden bağımsız iki gruptur.

İlişkisiz örneklemler t-test için gerekli varsayımlar analiz edilmiş ve tüm varsayımların karşılandığı belirlenmiştir. Tablo 5.10'da deney ve kontrol gruplarının Bilgisayar Programlamaya Karşı Tutum Ölçeği'nden almış oldukları notların t-test analiz sonuçları verilmiştir.

Tablo 5.10: Bilgisayar Programlamaya Karşı Tutum Ölçeği İlişkisiz Örneklemler t-Testi Sonuçları

Grup	**N**	**$\bar{X}$**	**Ss**	**sd**	**t**	**p**
Deney	22	3.91	0.44	40	.213	.825
Kontrol	20	3.88	0.41			

Tablo 5.10'da açıklanan sonuçlara göre deney ve kontrol gruplarının son test olarak verilmiş olan Bilgisayar Programlamaya Karşı Tutum Ölçeği'nden elde ettikleri puanlar ilişkisiz örneklemler t-testi ile incelendiğinde, deney grubunun ortalamasının kontrol grubuna göre daha yüksek olmasına rağmen aradaki bu farkın istatistiksel olarak anlamlı olmadığı tespit edilmiştir (t=0.223, p>.05). Bu sonuçlara bakıldığında hem GPÖ'nün hem de GEPÖ'nün öğrencilerin programlamaya karşı olumu bir tutum

sergilemelerinde etkili oldukları görülebilmektedir. Çünkü 5.00 üzerinden her iki grubun almış oldukları 3.91 ve 3.88 ortalama puanlar grupların olumlu tutuma sahip olduğunu göstermektedir (Tezbaşaran, 1997). Ayrıca GPÖ'nün öğrencilerin programlamaya karşı tutumlarında GEPÖ'ye kıyasla daha etkili olduğu ancak bu etkililiğin istatistiki olarak anlamlı olmadığı sonucuna ulaşılabilmektedir.

Ayrıca GPÖ ve GEPÖ'nün öğrencilerin programlamaya karşı tutumlarına etkisi arasında fark olup olmadığı BPKTÖ'ndeki alt boyutlar düzeyinde de incelenmiştir. Bunun için ilk olarak her bir alt boyut için Tablo 5.11'de de görüldüğü üzere Shapiro-Wilk normallik testi uygulanmıştır.

Tablo 5.11: Tutum Alt Boyutları İçin Shapiro-Wilk Normallik Testi

					Shapiro-Wilk		
	Grup	**N**	$\bar{X}$	**Ss**	**İstatistik**	**sd**	**p**
F1	**Deney**	22	3.35	0.70	.978	22	.887
	Kontrol	20	3.49	0.66	.968	20	.709
F2	**Deney**	22	4.32	0.45	.938	22	.176
	Kontrol	20	4.19	0.44	.987	20	.990
F3	**Deney**	22	4.40	0.44	.923	22	.087
	Kontrol	20	4.25	0.51	.945	20	.303
F4	**Deney**	22	4.37	0.61	.862	22	.005
	Kontrol	20	4.50	0.52	.760	20	.000

F1: Programlamada kendine güven ve güdülenme
F2: Programlamanın faydası
F3: Programlamada başarıya karşı tutum
F4: Programlamada başarının sosyal algısı

Tablo 5.11'deki sonuçlara göre ilk 3 boyuta air puanlar her iki grup için normal dağılırken, 4. boyuta ait puanlar normal dağılmamaktadır. Buna göre ilk 3 boyut için t-test, son boyut için Mann-Whitney U testi uygulanmıştır. T-teste ait sonuçlar Tablo 5.12'de yer alırken, Mann-Whitney U testine ait sonuçlar Tablo 5.13'te verilmiştir.

Tablo 5.12: Tutum Alt Boyutları için İlişkisiz Örneklemler T-testi

	Grup	N	$\bar{X}$	Ss	sd	t	p
F1	**Deney**	22	3.35	0.70	40	-.657	.515
	Kontrol	20	3.49	0.66			
F2	**Deney**	22	4.32	0.45	40	.955	.345
	Kontrol	20	4.19	0.44			
F3	**Deney**	22	4.40	0.44	40	.997	.325
	Kontrol	20	4.25	0.51			

Tablo 5.11'deki sonuçlara göre "*Programlamada kendine güven ve güdülenme"* alt boyutu için kontrol grubu lehine; *"Programlamanın faydası", "Programlamada başarıya karşı tutum"* alt boyutları için ise deney grubu lehine bir sonuç çıkarken, bu sonuçlar istatistiksel olarak anlamlı değildir.

Tablo 5.13: Tutum Alt Boyutu İçin Mann-Whitney U Testi

Grup	N	Sıra Ortalaması	Sıra Toplamı	Z	p
Deney	22	20.73	456.00	-.441	.659
Kontrol	20	22.35	447.00		

Tablo 5.13'teki sonuca göre *"Programlamada başarının sosyal algısı"* alt boyutu için kontrol grubu lehine istatistiksel olarak anlamlı olmayan bir sonuç çıkmıştır.

5.2 Çıkarımlar

Bu çalışma sonucunda aşağıdaki çıkarımlar yapılmıştır:

1. GPÖ'nün programlamayı öğrenme üzerine olan etkisi GEPÖ'ye göre anlamlı bir biçimde daha fazladır.

2. GPÖ ve GEPÖ öğrencilerin programlamaya karşı olumlu bir tutum sergilemelerine eşit seviyede katkıda bulunmuştur.

Sonuç olarak, bu çalışma görsel programlama öğretiminin öğrencilerin programlamayı öğrenmelerini ve programlamaya karşı tutumlarını olumlu bir biçimde etkilediğini göstermektedir.

BÖLÜM 6
TARTIŞMA VE ÖNERİLER

Bu bölümde çalışmadan elde edilen sonuçlara ilişkin tartışmalara ve çalışma ile ilgili önerilere yer verilmiştir.

6.1 Tartışma

Bu çalışma görsel programlama öğretiminin öğrencilerin programlamayı öğrenmelerine ve programlamaya karşı tutumlarına olan etkisini incelemek amacıyla yapılmıştır. Bunun için bir gruba görsel programlama öğretimine dayalı öğretim etkinlikleri ile programlama eğitimi verilmiş; diğer gruba da geleneksel programlama eğitimi verilmiştir. Grupların programlama başarı düzeylerini tahmin edebilmek ve grupları için çalışma başında PYT ön test olarak uygulanmış ve grupların test puanları arasında anlamlı bir fark olmadığı tespit edilmiştir. Bu da grupların çalışmanın başında programlamayı öğrenebilme seviyelerinin eşit olduklarını göstermektedir. Gruplar daha önce programlama dersi almadıklarından ön test olarak PKTÖ verilmemiştir. Çünkü öğrencilerin programlamaya karşı tutumunun çalışmanın sonunda oluşacağı beklenmektedir.

Çalışmanın 1. hipotezi olan "GPÖ ve GEPÖ'nün öğrencilerin programlamayı öğrenmesi üzerindeki etkilerinin arasında anlamlı bir fark yoktur" hipotezinin testi için deney ve kontrol grubuna son test olarak PBT uygulanmıştır. PYT'den alınan puanların ortak değişken olarak değerlendirilerek yapıldığı ANCOVA sonucunda GPÖ'nün öğrencilerin programlamayı öğrenmeleri üzerindeki etkisinin GEPÖ'ye göre anlamlı bir biçimde daha fazla olduğu bulgusuna ulaşılmıştır. Bu bulgu birçok araştırma tarafından da desteklenmektedir. Ozoron, Çağıltay ve Topallia (2012) üniversite öğrencileri üzerinde yaptıkları çalışmada Scratch kullanımının öğrencilerin programlama dersindeki başarılarını artırdığını belirlemişlerdir. Rizvi ve Humphries (2012) ise Scratch'ın üniversite öğrencilerinin programlamayı öğrenmedeki etkisini araştırdıkları çalışmada önceden Scratch eğitimi alan öğrencilerin almayanlara göre programlama dersini geçme yüzdelerinin daha yüksek olduğunu tespit etmiştir. Bishop-Clark ve diğ. (2007) bir başka görsel programlama ortamı olan Alice'in öğrencilerin temel programlama kavramlarını anlamalarına anlamlı bir biçimde katkı

sağladığı sonucuna varmışlardır. Bir başka çalışmada Sykes (2007) programlama dersinin belli bir kısmında Alice yazılımını kullanan grup ile hiç Alice kullanmayan grubun programlama başarılarını karşılaştırdığında, Alice'i kullanan grubun daha başarılı olduğunu gözlemlemiştir. Wang ve diğ. (2009) ise yaptıkları yarı deneysel çalışmada programlama dersini Alice üzerinden işleyen grubun, dersi C++ dili üzerinden işleyen gruba göre daha başarılı sonuçlar aldıklarını tespit etmişlerdir. GPÖ'nün programlama başarısına etkisini araştıran çalışma sayısı fazla olmamakla beraber, başarı ile ilişkisi olabilecek olan problem çözme becerisi üzerinde GPÖ'nün etkilerini inceleyen bazı çalışmalar da mevcuttur. Bu çalışmalar incelendiğinde GPÖ'nün öğrencilerin problem çözme becerilerini de geliştirdiği görülebilmektedir (Brown ve diğ., 2013; Shin ve Park, 2014).

GPÖ'nün programlama başarısına etkisi ile ilgili olarak bu çalışmadan elde edilen bulguyu desteklemeyen bazı çalışmalar da mevcuttur. Solmaz (2014) yaptığı çalışmada programlama dersinde Alice yazılımını kullanmanın programlama başarısı üzerinde anlamlı bir etki oluşturmadığını belirlemiştir. Mıhcı (2014) programlama dersinde bir görsel programlama aracı olan App Inventor yazılımını kullanmanın öğrencileri programlama başarısına anlamlı bir şekilde etki etmediğini tespit etmiştir. Kalelioğlu ve Gülbahar (2014) ise yaptıkları çalışmada Scratch'ın öğrencilerin problem çözme becerilerinde anlamlı bir artış sağlamadığı sonucuna varmışlardır.

Blockly kullanılarak uygulanan GPÖ'nün programlama öğrenimine anlamlı bir katkı yapmasının bazı nedenlerinin bulunduğu düşünülmektedir. Öncelikle Blockly'nin, diğer görsel programlama ortamlarında yapılan araştırmalarda da tespit edildiği gibi (Brown, 2008; Malan ve Leitner, 2007), programlama öğreniminde öğrencilerin hayli zorlandıkları ve birçok hata yaparak üzerinde fazla zaman harcadıkları programlama sözdizimini arka plana iterek daha önemli olarak görülen problem çözümünü ve algoritmik düşünmeyi desteklediğini söylemek mümkündür. Öğrenciler Blockly kullanarak program söz dizimi ile uğraşmamakta, direkt olarak problemin çözümü üzerine yoğunlaşmakta ve bunun üzerinde zaman harcamaktadırlar. Bunlara ek olarak, programlama öğrenimi açısından önemli olarak görülen tutum konusunda da, Blockly kullanan grubun programlamaya karşı olumlu bir tutum sergiledikleri görülmüş ve bu durumun da Blockly'nin programlama öğrenmeye katkı yapmış olabileceği düşünülmektedir.

Bu çalışmadan ve buna benzer çalışmalardan elde edilen bulgular göz önünde bulundurulduğunda GPÖ'nün öğrencilerin programlama başarılarını genel olarak olumlu bir biçimde etkilediği görülebilmektedir. Buradan Blockly'nin programlamaya giriş dersi için iyi bir öğretim aracı olduğu sonucu çıkartılabilmektedir.

Çalışmadaki 2. hipotez olan "GPÖ ve GEPÖ ile öğrenim gören öğrencilerin programlamaya karşı tutumları arasında anlamlı bir fark var yoktur " hipotezinin testi için her iki gruptaki öğrencilere son test olarak BPKTÖ uygulanmıştır. Yapılan analiz sonucunda GPÖ ile öğrenim gören öğrencilerin programlamaya karşı tutumları GEPÖ ile öğrenim gören öğrencilere göre daha yüksek olmasına rağmen aradaki bu farkın istatistiksel olarak anlamlı olmadığı tespit edilmiştir. Bununla beraber iki öğretim metodunun da öğrencilerin programlamaya karşı olumlu tutum sergilemelerini sağladığı görülmüştür. Alanyazın incelendiğinde bu çalışmada elde edilen bulguya benzer olarak GPÖ'nün programlamaya karşı olumlu tutum sağladığını belirten çalışmalar mevcuttur. Kalelioğlu ve Gülbahar (2014) yaptıkları çalışmada Scratch'ın programlama dersinde kullanımının derse karşı tutumu iyileştirdiğini gözlemlemişlerdir. Öğretmen adayları üzerinde yapılan çalışmada ise programlama dersinde Scratch yazılımının öğrencilerin ders hakkındaki tutumlarını olumlu olarak etkilediği sonucuna varılmıştır (Fesakis ve Serafeim, 2009). Bunlara ek olarak, bu çalışmada olduğu gibi GPÖ ve GEPÖ'yü karşılaştıran ve bu iki yöntemin programlamaya karşı tutumlara etkisi noktasında anlamlı bir fark olmadığını tespit eden çalışmalar da mevcuttur. Korkmaz (2016) çalışmasında Scratch'ın öğrencilerin programlamaya karşı tutumları üzerinde anlamlı bir etki oluşturmadığını belirtmiştir. Hutchinson, Moskal, Dann ve Cooper (2008) ise benzer bir sonuç elde etmiş ve Alice kullanımının programlamaya karşı tutum üzerinde anlamlı bir etkisinin olmadığını ifade etmişlerdir.

GPÖ'nün GEPÖ'ye göre tutum konusunda anlamlı bir etkisinin olmamasının altında bir takım sebepler bulunabilir. Deney grubunda Blockly ile ilgili olumlu düşüncelere sahip öğrenciler olabileceği gibi, Blockly'nin Python ya da C gibi gerçek bir programlama dili olmadığını ve kontrol grubundaki öğrenciler gibi metinsel olarak programlama öğrenmenin daha yararlı olacağını düşünen öğrencilerin de bulunma ihtimali vardır. Nitekim bu konuda yapılmış araştırmalarda görsel programlama ortamlarını basit bulan ve gerçekçi olmadığını düşünen öğrencilerin bulunabileceği

tespit edilmiştir (Kasurinen, Purmonen ve Nikula, 2008; Mullins, Whitfield ve Conlon, 2009; Tanrıkulu ve Schaefer, 2011**).** Bu durum GPÖ'nün programlamaya karşı tutum için anlamlı bir etkisinin olamamasına sebep olarak gösterilebilir.

Alanyazında da görüleceği üzere programlama öğreniminin bir takım zorlukları bulunmakta ve bu soruna yönelik olarak görsel teknolojilerden yararlanılabileceği ifade edilmektedir (Adams, 2014; Herbert, 2010). Bu çalışmada kullanılan Blockly yazılımının programlama öğrenimindeki zorlukları ne derece giderebildiği ise tartışmaya değer bir konu olarak görülmektedir. İlk olarak, programlama öğreniminin en önde gelen zorluklarından birisi programlama için sahip olunması gereken bilgi çeşididir (sözdizimsel, kavramsal ve stratejik bilgi) (Mannila, Peltomäki ve Salakoski, 2006). Alanyazında bu durumun programlama öğrenmeyi yeni öğrenenler için zor hale getirdiği ve kavramsal ve stratejik bilginin geliştirilmesi yerine daha çok sözdizimsel bilgi üzerinde vakit harcandığından bahsedilmektedir (Jenkins ve Davy, 2000; McGill ve Volet, 1997). Blockly'nin ise bu noktada devreye girdiğini ve sözdizimini arka planda tutarak, öğrencilerin programlama kavramları ve stratejik düşünmelerini kolaylaştırdığı söyleyenebilir. Bu özelliği ile Blockly'nin programlamadaki bahsedilen bu zorluğu gidermekte etkili olduğu varsayılabilir.

İkinci olarak, kullanılan programlama türü ve programlama dilinin de programlamayı öğrenmeyi zorlaştırabileceği belirtilmiştir (Goldwasser ve Letscher, 2008; Vujoševic-Janicic ve Tošic, 2008). Alanyazında programlamaya giriş derslerinde nesne-tabanlı öğretim tarzı yerine işleve dayalı programlama öğretim tarzının benimsenmesinin yararlı olacağı ifade edilmiştir (Cooper, Dann ve Pausch, 2003; Reges, 2006). Ayrıca programlama dili tercihi olarak da Python programlama dilinin giriş derslerinde daha uygun olacağından söz edilmiştir (Fangohr, 2004; Oldham, 2005; Sanders ve Langford, 2008). Bu bağlamda, Blockly'nin hem işleve dayalı bir programlama öğretim tarzını benimsemiş olduğu hem de araştırmacılar tarafından Python programlama diline uygun olarak geliştirilmiş olduğu göz önünde bulundurulduğunda, Blockly'nin programlamadaki programlama türü ve dilinden kaynaklı olabilecek zorluğu da giderebilmesinden söz etmek mümkün görülmektedir.

Programlamaya karşı sahip olunan olumsuz tutumun da programlama öğrenimini zorlaştırabileceğinden bahsedilmiştir (Anastasiadou ve Karakos, 2011; Farkas ve Murthy, 2005; Korkmaz ve Altun, 2013). Bu çalışmada elde edilen veriler ışığında,

Blockly'nin öğrencilerin programlamaya karşı olumlu tutum sergilemelerini sağladığı tespit edilmiştir. Böylelikle Blockly'nin programlamayı öğrenme zorluklarından birisi olan tutumla ilgili problemlerin düzeltilmesinde de rol oynadığı düşünülmektedir.

Özet olarak, bu çalışmada GPÖ'nün öğrencilerin programlamayı öğrenmelerine ve programlamaya karşı tutumlarına etkisi incelenmiştir. Çalışma sonucunda GPÖ'nün programlamayı öğrenme konusunda GEPÖ'ye göre daha etkili olduğu tespit edilmiştir. GPÖ'nün ayrıca öğrencilerin programlamaya karşı olumlu tutuma sahip olmasında da etkili olduğu, ancak bunun GEPÖ'ye kıyasla anlamlı bir biçimde farklılaşmadığı gözlenmiştir. Ayrıca Blockly kullanımının programlamadaki bazı öğrenme zorluklarını gidermede de etkili olduğu sonucuna ulaşılmıştır.

6.2 Öneriler

Bu çalışmadan elde edilen bulgulara bağlı olarak aşağıdaki öneriler verilmiştir:

1. GPÖ yaklaşımının Blockly yazılımı üzerinden farklı üniversitelerdeki Programlamaya Giriş derslerindeki etkisi araştırılabilir.

2. Blockly yazılımının daha sadeleştirilerek ya da geliştirilerek farklı öğrenim düzeylerinde programlama öğrenimi gören öğrenciler üzerindeki etkisi araştırılabilir.

3. Blockly yazılımının Python'dan farklı programlama dilleri ile birlikte yürütüldüğü Programlama derslerindeki etkisi araştırılabilir.

4. Blockly'nin metin tabanlı programlama dillerine geçişteki etkisini araştıran çalışmalar yapılabilir.

5. Blockly yazılımının zihinsel yükü azaltma noktasında etkili olup olmadığını araştıran çalışmalar yapılabilir.

KAYNAKÇA

Adams, J. (2014). *Alice 3 in Action: Computing Through Animation*. Cengage Learning.

Agarwal, K. K., Agarwal, A., & Celebi, M. E. (2008). Python puts a squeeze on Java for CS0 and beyond. *Journal of Computing Sciences in Colleges, 23*(6), 49-57.

Akpınar, Y., Altun, A. (2014). Bilgi toplumu okullarında programlama eğitimi gereksinimi. *Elementary Education Online, 13(1),* 1-4.

Allert, J. (2004). *Learning Style and Factors Contributing to Success in an Introductory Computer Science Course*. Paper presented at Fourth IEEE International Conference on Advanced Learning Technologies (ICALT'04).

Ambrósio, A. P., Costa, F. M., Almeida, L., Franco, A., & Macedo, J. (2011). *Identifying cognitive abilities to improve CS1 outcome.* Paper presented at *Frontiers in Education Conference.*

Anastasiadou, S. D., & Karakos, A. S. (2011). The Beliefs of Electrical and Computer Engineering Students' Regarding Computer Programming. *International Journal of Technology, Knowledge & Society*, *7*(1), 37-51.

Bandura, A. (1977). Self-efficacy: toward a unifying theory of behavioral change. *Psychological review*, *84*(2), 191-215.

Bati, T. B., Gelderblom, H., & van Biljon, J. (2014). A blended learning approach for teaching computer programming: design for large classes in Sub-Saharan Africa. *Computer Science Education*, *24*(1), 71-99

Başer, M. (2013a). Bilgisayar Programlamaya Karşı Tutum Ölçeği Geliştirme Çalışması. *The Journal of Academic Social Science Studies*, *6*(6), 199–215.

Başer, M. (2013b). Attitude, Gender and Achievement in Computer Programming. *Middle-East Journal of ScientificResearch, 14*(2), 248-255.

Başer, M. (2013c). *Python*. Dikey Eksen, İstanbul.

Bayman, P. ve Mayer, R., (1988), Using Conceptual Models to Teach BASIC Computer Programming, *Journal of Educational Psychology, 80(3),* 291-298.

Ben-Bassat Levy, R., Ben-Ari, M., & Uronen, P. A. (2003). The Jeliot 2000 program animation system. *Computers & Education*, *40*(1), 1-15.

Bennedsen, J.,& Caspersen, M. E. (2007). Failure rates in introductory programming. *ACM SIGCSE Bulletin*, *39*(2), 32-36.

Bergin, S., & Reilly, R. (2006). Predicting introductory programming performance: A multi-institutional multivariate study. *Computer Science Education*, *16*(4), 303-323.

Bishop-Clark, C., Courte, J., Evans, D., & Howard, E. V. (2007). A quantitative and qualitative investigation of using Alice programming to improve confidence, enjoyment and achievement among non-majors. *Journal of Educational Computing Research*, *37*(2), 193-207.

Blockly (2015). https://blockly-demo.appspot.com/static/demos/code/index.html. Erişim tarihi: 13.01.2015.

Bornat, R., & Dehnadi, S. (2008). Mental models, consistency and programming aptitude. In *Proceedings of the tenth conference on Australasian computing education* (pp. 53-61).

Böszörményi, L. (1998). Why Java is not my favorite first-course language. *Software-Concepts & Tools*, *19*(3), 141-145.

Brown, P. H. (2008). Some field experience with Alice. *Journal of Computing Sciences in Colleges*, *24*(2), 213-219.

Brown, Q., Mongan, W., Kusic, D., Garbarine, E., Fromm, E., & Fontecchio, A. (2013). Computer aided instruction as a vehicle for problem solving: Scratch programming environment in the middle years classroom. *Retrieved September*, *22*.

Bruce, K. B. (2005). Controversy on how to teach CS 1: a discussion on the SIGCSE-members mailing list. *ACM SIGCSE Bulletin*, *37*(2), 111-117.

Büyüköztürk, Ş. (2014). *Veri Analizi El Kitabı*. Pegem Akademi, Ankara.

Büyüköztürk, Ş., Çakmak, E. K., Akgün, Ö. E., & Karadeniz, Ş. ve Demirel, F.(2008). *Bilimsel araştırma yöntemleri*. Pegem Akademi, Ankara.

Byrne, P., & Lyons, G. (2001). The effect of student attributes on success in programming. *ACM SIGCSE Bulletin, 33(3),* 49-52

Casey, P. J. (1997). Computer programming: A medium for teaching problem solving. *Computers in the Schools*, *13*(1-2), 41-51.

Caspersen, M. E., Larsen, K. D., & Bennedsen, J. (2007). Mental models and programming aptitude. *ACM SIGCSE Bulletin, 39(3),* 206-210.

Chen, L.,& Lin, J.M. (2011). Learning styles and student performance in java programming courses. *Proceedings of the 2011 international conference on frontiers in education: CS and CE*, 53-58.

Clark, R. C., & Lyons, C. (2010). *Graphics for learning: Proven guidelines for planning, designing, and evaluating visuals in training materials*. John Wiley & Sons.

Computer Programming Aptitude Test (2015). http://www.kent.ac.uk/careers/tests/computer-test.htm. Erişim tarihi: 29.07.2015.

Cooper, S., Dann, W., & Pausch, R. (2000a). Alice: a 3-D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges*, *15(5),* 107-116.

Cooper, S., Dann, W., & Pausch, R. (2000b). Developing algorithmic thinking with Alice. In *The Proceedings of ISECON 2000, 17,* 506-539.

Cooper, S., Dann, W., & Pausch, R. (2003). Teaching objects-first in introductory computer science. *ACM SIGCSE Bulletin*, *35*(1), 191-195.

Corney, M., Teague, D., & Thomas, R. N. (2010). Engaging students in programming. In *Proceedings of the 12th Australasian Conference on Computing Education* (pp. 63-72).

Corney, M., Lister, R., & Teague, D. (2011). Early relational reasoning and the novice programmer: swapping as the hello world of relational reasoning. In *Proceedings of the Thirteenth Australasian Computing Education Conference-Volume* (pp. 95-104).

Courte, J., Howard, E., & Bishop-Clark, C. (2006). Using Alice in a computer science survey course. *Information Systems Education Journal*, *4*, 1-7.

Çakıroğlu, Ü., Sarı, E. & Akkan, Y. (2011). The View of the Teachers about the Contribution of Teaching Programming to the Gifted Students in the Problem Solving. *In 5th International Computer & Instructional Technologies Symposium. Fırat University.*

Dann, W., Cooper, S., & Pausch, R. (2000). Making the connection: programming with animated small world. *ACM SIGCSE Bulletin, 32(3),* 41-44.

Dann, W., Cosgrove, D., Slater, D., Culyba, D., & Cooper, S. (2012). Mediated transfer: Alice 3 to java. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (pp. 141-146).

Dehnadi, S. (2006). Testing programming aptitude. In *Proceedings of the 18th Annual Workshop of the Psychology of Programming Interest Group* (pp. 22-37).

Ebrahimi, A. (1994). Novice programmer errors: Language constructs and plan composition. *International Journal of Human-Computer Studies*, *41*(4), 457-480.

Eckerdal, A., McCartney, R., Moström, J. E., Ratcliffe, M., & Zander, C. (2006). Can graduating students design software systems?. *ACM SIGCSE Bulletin*, *38*(1), 403-407.

Edirisinghe, E. M. (2008). Teaching students to identify common programming errors using a game. In *Proceedings of the 9th ACM SIGITE conference on Information technology education* (pp. 95-98).

Everitt, B.S., & Skrondal, A. (2010). *The Cambridge Dictionary of Statistics (Fourth Edition).* Cambridge: Cambridge University Press.

Eymur, G., & Geban, Ö. (2011). Kimya öğretmeni adaylarının motivasyon ve akademik başarıları arasındaki ilişkinin incelenmesi. *Eğitim ve Bilim*, *36*(161), 246-255.

Fangohr, H. (2004). A comparison of C, MATLAB, and Python as teaching languages in engineering. In *Computational Science-ICCS 2004* (pp. 1210-1217).

Farkas, D., & Murthy, N. (2005). Attitudes toward computers, the introductory course and recruiting new majors: Preliminary results. In *17th Workshop of the Psychology of Programming Interest Group, Sussex University* (pp. 268-277).

Fesakis, G., & Serafeim, K. (2009). Influence of the familiarization with scratch on future teachers' opinions and attitudes about programming and ICT in education. *ACM SIGCSE Bulletin, 41(3),* 258-262.

Field, A. (2013). *Discovering statistics using IBM SPSS statistics*. Sage.

Fincher, S., Robins, A., Baker, B., Box, I., Cutts, Q., de Raadt, M., ... & Petre, M. (2006). Predictors of success in a first programming course. In *Proceedings of the 8th Australasian Conference on Computing Education* (pp. 189-196).

Fowler, L., Allen, M., Armarego, J., Mackenzie, J. (2000). Learning styles and CASE tools in software engineering. In *Proceedings of the 9th Annual Teaching Learning Forum* (pp. 1-10).

Ghaith, G. M. (2003). Relationship between reading attitudes, achievement, and learners perceptions of their Jigsaw II cooperative learning experience. *Reading Psychology*, *24*(2), 105-121.

Gill, T. G. and Holton, C. F. (2006). A self-paced introductory programming course. *Journal of Information Technology Education*, 5, 95-105.

Gobil, A., Shukor, Z., & Mohtar, I. A. (2009). Novice difficulties in selection structure. *International Conference on Electrical Engineering and Informatics, 2,* 351-356.

Golding, P., Facey-Shaw, L., & Tennant, V. (2006). Effects of peer tutoring, attitude and personality on academic performance of first year introductory programming students. In *Frontiers in Education Conference, 36th Annual* (pp. 7-12). IEEE.

Goldwasser, M. H.,& Letscher, D. (2008). Teaching an object-oriented CS1 with Python. *ACM SIGCSE Bulletin, 40(3),* 42-46.

Gomes, A., Carmo, L., Bigotte, E., & Mendes, A. (2006). Mathematics and programming problem solving. In *3rd E-Learning Conference–Computer Science Education*.

Gomes, A.,& Mendes, A. J. (2007). Learning to program-difficulties and solutions. In *International Conference on Engineering Education–ICEE*.

Gomes, A. J.,& Mendes, A. J. (2010). A study on student performance in first year CS courses. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education* (pp. 113-117).

Goodman, S., Jaffer, T., Keresztesi, M., Mamdani, F., Mokgatle, D., Musariri, M., ... & Schlechter, A. (2011). An investigation of the relationship between students' motivation and academic performance as mediated by effort. *South African Journal of Psychology*, *41*(3), 373-385.

Goosen, G. L., Mentz, E., & Nieuwoudt, E. (2007). Choosing the "Best" Programming Language. In *Proceedings of the Computer Science and IT Education Conference* (pp. 269-282).

Grandell, L., Peltomäki, M., & Salakoski, T. (2005). High school programming—a beyond-syntax analysis of novice programmers' difficulties. In *Proceedings of the Koli Calling 2005 Conference on Computer Science Education* (pp. 17-24).

Grandell, L., Peltomäki, M., Back, R. J., & Salakoski, T. (2006). Why complicate things?:introducing programming in high school using Python. In *Proceedings of the 8th Australasian Conference on Computing Education* (pp. 71-80).

Hagan, D., & Markham, S. (2000). Does it help to have some programming experience before beginning a computing degree program?. *ACM SIGCSE Bulletin, 32(3),* 25-28.

Hair, J.F., Black, W.C., Babin, B.J., & Anderson, R.E. (2010) . *Multivariate Data Analysis* . Seventh Edition. Prentice Hall, Upper Saddle River, New Jersey.

Harris, J. (2014). Testing programming aptitude in introductory programming courses. *Journal of Computing Sciences in Colleges*, *30*(2), 149-156.

Herbert, C. W. (2010). *An introduction to programming using Alice 2.2*. Cengage Learning.

Hu, C. (2004). Rethinking of teaching objects-first. Education *and Information technologies, 9(3),* 209-218.

Hutchinson, A., Moskal, B., Dann, W., & Cooper, S. (2008). Impact of the Alice curriculum on community college students' attitudes and learning with respect to computer science. In *Paper appears in the proceedings of the annual meeting of the American Society for Engineering Education, Pittsburgh, PA.*

Jenkins, T. (2002). On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences* (pp. 53-58).

Jenkins, T., & Davy, J. (2000). Dealing with diversity in introductory programming. *In Proceedings of 1st Annual LTSN-ICS Conference*, (pp. 81-87).

Jha, A. S. (2014). *Social research methods*. Tata McGraw-Hill Education.

Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students: Code. org. *Computers in Human Behavior*, *52*, 200-210.

Kalelioglu, F., & Gülbahar, Y. (2014). The effects of teaching programming via scratch on problem solving skills: A discussion from learners' perspective. *Informatics in Education*, *13*(1), 33-50.

Kannusmäki, O., Moreno, A., Myller, N., & Sutinen, E. (2004). What a novice wants: students using program visualization in distance programming course. In *Proceedings of the Third Program Visualization Workshop* (pp. 126-133).

Kasurinen, J., Purmonen, M., & Nikula, U. (2008). A study of visualization in introductory programming. In *Proceedings of the Annual Meeting of the Psychology of Programming Interest Group* (pp. 181-194).

Kaucic, B.,& Asic, T. (2011). Improving introductory programming with Scratch?. In *MIPRO, 2011 Proceedings of the 34th International Convention* (pp. 1095-1100).

Kelleher, C.,& Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR)*, *37*(2), 83-137.

Kinnunen, P., & Malmi, L. (2006). Why students drop out CS1 course?. In *Proceedings of the second international workshop on Computing education research* (pp. 97-108).

Kirby, S., Toland, B., & Deegan, C. (2010). Program visualization tool for teaching programming in C. In *Proceedings of the International Conference on Education, Training and Informatics (ICETI'10)*.

Kordaki, M. (2012). Diverse categories of programming learning activities could be performed within Scratch. *Procedia-Social and Behavioral Sciences*, *46*, 1162-1166.

Korkmaz, Ö. (2013). Prospective CITE Teachers' Self-efficacy Perceptions on Programming. *Procedia-Social and Behavioral Sciences, 83,* 639-643.

Korkmaz, Ö. (2016). The Effects of Scratch-Based Game Activities on Students' Attitudes, Self-Efficacy and Academic Achievement. *education, 3*(4), 13-15.

Korkmaz, Ö., & Altun, H. (2013). Engineering and ceit student's attitude towards learning computer programming. *The Journal of Academic Social Science Studies International Journal of Social Science*, *6*(2), 1169-1185.

Krpan, D., Mladenović, S., & Rosić, M. (2015). Undergraduate programming courses, students' perception and success. *Procedia-Social and Behavioral Sciences, 174*, 3868-3872.

Laakso, T. R. M. J., Kaila, E., & Salakoski, T. (2008). Effectiveness of program visualization: A case study with the ViLLE tool. *Journal of Information Technology Education*, *7*, 15-32.

Lacher, L. L., & Lewis, M. C. (2015). The Effectiveness of Video Quizzes in a Flipped Class. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 224-228).

Lahtinen, E., Ala-Mutka, K., & Järvinen, H.-M. (2005). A Study of the Difficulties of Novice Programmers. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (pp. 14-18).

Lane, H. C.,& VanLehn, K. (2005). Teaching the tacit knowledge of programming to novices with natural language tutoring. *Computer Science Education, 15*(3), 183–201.

Lau, W. W., & Yuen, A. H. (2009). Exploring the effects of gender and learning styles on computer programming performance: implications for programming pedagogy. *British Journal of Educational Technology*, *40*(4), 696-712.

Lavonen, J. M., Meisalo, V. P., Lattu, M., & Sutinen, E. (2003). Concretising the programming task: a case study in a secondary school. *Computers & Education*, *40*(2), 115-135.

Layman, L., Cornwell, T., & Williams, L. (2006). Personality types, learning styles, and an agile approach to software engineering education. *ACM SIGCSE Bulletin*, *38*(1), 428-432.

Linn, M. C. (1985). The cognitive consequences of programming instruction in classrooms. *Educational Researcher*, *14*(5), 14-29.

Lister, R. (2001). Objectives and objective assessment in CS1. *ACM SIGCSE Bulletin, 33(1),* 292-296.

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., ... & Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin*, *36*(4), 119-150.

Lister, R., Fidge, C., & Teague, D. (2009). Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *ACM SIGCSE Bulletin*, *41*(3), 161-165.

Loftus, C., Thomas, L., & Zander, C. (2011). Can graduating students design: revisited. In *Proceedings of the 42nd ACM technical symposium on Computer science education* (pp. 105-110). ACM.

Lopez, M., Sutton, K., & Clear, T. (2009). Surely we must learn to read before we learn to write!. In *Proceedings of the Eleventh Australasian Conference on Computing Education (pp.* 165-170).

Madison, S.,& Gifford, J. (2002). Modular programming: novice misconceptions. *Journal of Research on Technology in Education*, *34*(3), 217-229.

Malan, D. J.,& Leitner, H. H. (2007). Scratch for budding computer scientists. *ACM SIGCSE Bulletin, 39(1),* 223-227.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE), 10*(4), 1-5.

Mannila, L., Peltomäki, M., & Salakoski, T. (2006). What about a simple language? Analyzing the difficulties in learning to program. *Computer Science Education*, *16*(3), 211-227.

Maragos, K., & Grigoriadou, M. (2011). Exploiting TALENT as a Tool for Teaching and Learning. *International Journal of Learning*, *18*(1), 431-440.

Martinez, R. S., Aricak, O. T., & Jewell, J. (2008). Influence of reading attitude on reading achievement: A test of the temporal-interaction model. *Psychology in the Schools*, *45*(10), 1010-1023.

Mazman, S.G. (2013). *Programlama performansını etkileyen faktörlerin bilişsel tabanlı bireysel farklılıklar temelinde modellenmesi.* (Yayımlanmamış doktora tezi). Hacettepe Üniversitesi, Ankara.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B. D., ... & Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*, *33*(4), 125-180.

McGill, T. J., & Volet, S. E. (1997). A conceptual framework for analyzing students' knowledge of programming. *Journal of research on Computing in Education, 29(3),* 276-297.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2011). Habits of programming in Scratch. In *Proceedings of the Sixteenth SIGCSE Conference on Innovation and Technology in Computer Science Education, Darmstadt, Germany* , (pp. 168-172).

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with scratch. *Computer Science Education*, *23*(3), 239-264.

Mehic, N., Hasan, Y., & KPMG, B. (2001). Challenges in teaching java technology. *Informing Science*, 365-371.

Mhashi, M. M.,& Alakeel, A. (2013). Difficulties Facing Students in Learning Computer Programming Skills at Tabuk University. In *Proceedings of the 12th International Conference on Education and Educational Technology* (pp. 15-24).

Mıhcı, C. (2014). *Programlama eğitiminde görsel blok programlama ve mobil uygulama geliştirme araçlarının karşılaştırılması.* (Yayımlanmamış yüksek lisans tezi). Marmara Üniversitesi, İstanbul.

Miller, M. D., Linn, R. L., Gronlund, N.E. (2008). *Measurement and assessment in teaching*. Pearson Education India.

Miller, B., & Ranum, D. (2006). Freedom to succeed: a three course introductory sequence using Python and Java. *Journal of Computing Sciences in Colleges*, *22*(1), 106-116.

Moons, J.,& De Backer, C. (2013). The design and pilot evaluation of an interactive learning environment for introductory programming influenced by cognitive load theory and constructivism. *Computers & Education*, *60*(1), 368-384.

Moreno, A.,& Myller, N. (2003). Producing an educationally effective and usable tool for learning, the case of the Jeliot family. In *the Proceedings of International Conference on Networked e-learning for European Universities. Granada, Spain*.

Moreno, A., Myller, N., & Bednarik, R. (2005). Jeliot 3, an extensible tool for program visualization. In *5th Annual Finnish/Baltic Sea Conference on Computer Science Education*.

Moskal, B., Lurie, D., & Cooper, S. (2004). Evaluating the effectiveness of a new instructional approach. *ACM SIGCSE Bulletin, 36(1),* 75-79.

Mow, I. T. C. (2012). Analyses of Student Programming Errors In Java Programming Courses. *Journal of Emerging Trends in Computing and Information Sciences*, 3(5), 739-749.

Mullins, P., Whitfield, D., & Conlon, M. (2009). Using Alice 2.0 as a first language. *Journal of Computing Sciences in Colleges*, *24*(3), 136-143.

Muratet, M., Torguet, P., Viallet, F., & Jessel, J. P. (2011). Experimental feedback on Prog&Play: a serious game for programming practice. *Computer Graphics Forum, 30(1),* 61-73.

Müller, M. M. (2007). Do programmer pairs make different mistakes than solo programmers?. *Journal of Systems and Software*, *80*(9), 1460-1471.

Myers, B. A. (1990). Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing*, *1*(1), 97-123.

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., ... &Velázquez-Iturbide, J. Á. (2002). Exploring the role of visualization and engagement in computer science education. *ACM SIGCSE Bulletin, 35(2),* 131-152.

Nikula, U., Sajaniemi, J., Tedre, M., & Wray, S. (2007). Python and roles of variables in introductory programming: experiences from three educational institutions. *Journal of Information Technology Education: Research*, *6*(1), 199-214.

Oldham, J. D. (2005). What happens after Python in CS1?. *Journal of computing sciences in colleges, 20*(6), 7-13.

Ozorana, D., Cagiltay, N. E., & Topallia, D. (2012). Using Scratch in introduction to programming Course for Engineering Students. *MEUK2012, Antalya, Turkey***.**

Özyurt, Ö.,& Özyurt, H. (2015). A study for determining computer programming students' attitudes towards programming and theirprogramming self-efficacy. *Journal of Theory and Practice in Education, 11(1),* 51-67.

Paivio, A. (1990). *Mental representations. A dual coding approach* . New York: Oxford University Press.

Paliokas, I., Arapidis, C., & Mpimpitsos, M. (2011). PlayLOGO 3D: a 3D interactive video game for early programming education: let LOGO be a game. In *Games and Virtual Worlds for Serious Applications (VS-GAMES), 2011 Third International Conference* (pp. 24-31). IEEE.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc.

Patterson-McNeill, H. (2006). Experience: from C++ to Python in 3 easy steps. *Journal of Computing Sciences in Colleges*, *22*(2), 92-96.

Pea, R. D., & Kurland, D. M. (1983). On the cognitive prerequisites of learning computer programming.

Pillay, N. & Jugoo, V. (2005). An investigation into student characteristics affecting novice programming performance. *ACM SIGCSE Bulletin 37*, 107-110.

Polo, B. J. (2013). *SBL-online: Implementing studio-based learning techniques in an online introductory programming course to address common programming errors and misconceptions.* (Yayımlanmamış Doktora Tezi, University of Hawaii).

Powers, K., Gross, P., Cooper, S., McNally, M., Goldman, K. J., Proulx, V., & Carlisle, M. (2006). Tools for teaching introductory programming: what works?. *ACM SIGCSE Bulletin, 38(1),* 560-561.

Proulx, V. K., Raab, J., & Rasala, R. (2002). Objects from the beginning-with GUIs. *ACM SIGCSE Bulletin*, *34*(3), 65-69.

Ramalingam, V., LaBelle, D., & Wiedenbeck, S. (2004). Self-efficacy and mental models in learning to program. In *ACM SIGCSE Bulletin, 36(3),* 171-175.

Reges, S. (2006). Back to basics in CS1 and CS2. *ACM SIGCSE Bulletin, 38*(1), 293-297.

Rizvi, M.,& Humphries, T. (2012). A Scratch-based CS0 course for at-risk computer science majors. In *Frontiers in Education Conference (FIE), 2012*, 1-15.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education, 13*(2), 137-172.

Rogozhkina, I., & Kushnirenko, A. (2011). PiktoMir: teaching programming concepts to preschoolers with a new tutorial environment. *Procedia-Social and Behavioral Sciences, 28*, 601-605.

Rountree, N., Rountree, J., Robins, A., & Hannah, R. (2004). Interacting factors that predict success and failure in a CS1 course. *ACM SIGCSE Bulletin, 36(4),* 101-104.

Sak, N., & Demirer, V. (2016). Programming education and new approaches around the world and in Turkey. *Journal of Theory and Practice in Education, 12(3), 521-546.*

Sanders, I. D., & Langford, S. (2008). Students' perceptions of python as a first programming language at wits. *ACM SIGCSE Bulletin, 40(3),* 365-365.

Scott, A. S. (2010). *Using flowcharts, code and animation for improved comprehension and ability in novice programming* (Yayımlanmamış Doktora Tezi, University of Glamorgan).

Šerbec, I. N., Kaučič, B., & Rugelj, J. (2008). Pair Programming as a Modern Method of Teaching Computer Science. *International Journal of Emerging Technologies in Learning*, *3*, 45-49.

Shin, S., Park, P. (2014). A Study on the Effect affecting Problem Solving Ability of Primary Students through the Scratch Programming. *Advanced Science and Technology Letters, 59*, 117-120.

Shu, N. C. (1999). Visual programming: Perspectives and approaches. *IBM Systems Journal*, *38*(4), 199-221.

Sims, R.R., and Sims S.J. (1995). *The Importance of Learning Styles: Understanding the Implications For Learning Course Design and Education*. Westport CT: Greenwood Press.

Solmaz, E. (2014). *Programlama dili öğretiminde Alice yazılımının ders başarısı, eleştirel düşünme ve problem çözme becerileri ile üstbilişsel farkındalık düzeyine etkisi*. (Yayımlanmamış doktora tezi). Gazi Üniversitesi, Ankara.

Soloway, E.,& Ehrlich, K. (1984). Empirical Studies of Programming Knowledge. *IEEE Transactions on Software Engineering*, *10*(5), 595-609.

Sorge, C., & Schau, C. (2002). Impact of engineering students' attitudes on achievement in statistics: A structural model. In *annual meeting of the American Educational Research Association. New Orleans*.

Sorva, J. (2012). *Visual program simulation in introductory programming education. (*Yayımlanmamış Doktora Tezi). Aalto University at Finland.

Sorva, J., Lönnberg, J., & Malmi, L. (2013). Students' ways of experiencing visual program simulation. *Computer Science Education*, *23*(3), 207-238.

Sykes, E. R. (2007). Determining the effectiveness of the 3D Alice programming e nvironment at the computer science I level. *Journal of Educational Computing Research*, *36*(2), 223-244.

Tai, D. W., Yu, C. H., Lai, L. C., & Lin, S. J. (2003). A study on the effects of spatial ability in promoting the logical thinking abilities of students with regard to programming language. *World Transactions on Engineering and Technology Education UICEE, 2*(2), 251-26.

Talton, J. O., Peterson, D. L., Kamin, S., Israel, D., & Al-Muhtadi, J. (2006). Scavenger hunt: computer science retention through orientation. *ACM SIGCSE Bulletin, 38(1)*, 443-447.

Tanrikulu, E., & Schaefer, B. C. (2011). The users who touched the ceiling of scratch. *Procedia-Social and Behavioral Sciences, 28*, 764-769.

Teague, D. M., & Roe, P. (2009). Learning to program: from pear-shaped to pairs. In *Proceedings of the First International Conference on Computer Supported Education* (pp. 151-158).

Tella, A. (2007). The impact of motivation on student's academic achievement and learning outcomes in mathematics among secondary school students in Nigeria. *Eurasia Journal of Mathematics, Science & Technology Education, 3*(2), 149-156.

Tezbaşaran, A. A. (1997). Likert tipi ölçek geliştirme kılavuzu. *Ankara: Türk Psikologlar Derneği Yayınları, 12*, 22-25.

Thomas, L., Ratcliffe, M., Woodbury, J., & Jarman, E. (2002). Learning styles and performance in the introductory programming sequence. *ACM SIGCSE Bulletin,34(1*), 33-37.

Tuckman, B. W., & Harper, B. E. (2012). *Conducting educational research*. Rowman & Littlefield Publishers.

Tukiainen, M., & Mönkkönen, E. (2002). Programming aptitude testing as a prediction of learning to program. In *Proc. 14th Workshop of the Psychology of Programming Interest Group* (pp. 45-57).

Van-Roy, P., & Haridi, S. (2004). *Concepts, techniques, and models of computer programming*. MIT press.

Ventura, P. R. (2005). Identifying predictors of success for an objects-first CS1. *Computer Science Education, 15*, 223-243.

Vegso, J. (2005). Interest in CS as a major drops among incoming freshmen. *Computing Research News, 17*(3), 6-1.

Vujoševic-Janicic, M., & Tošic, D. (2008). The role of programming paradigms in the first programming courses. *The Teaching of Mathematics, XI, 2*, 63-83.

Wang, T. C., Mei, W. H., Lin, S. L., Chiu, S. K., & Lin, J. C. (2009). Teaching programming concepts to high school students with alice. In *Frontiers in Education Conference* (pp. 1-6).

Weintrop, D., & Wilensky, U. (2015). To Block or Not to Block, That is the Question: Students' Perceptions of Blocks-based Programming. In *Proceedings of the 14th International Conference on Interaction Design and Children* (pp. 199–208).

Wiedenbeck, S., Labelle, D., & Kain, V. N. (2004). Factors affecting course outcomes in introductory programming. In *16th Annual Workshop of the Psychology of Programming Interest Group* (pp. 97-109).

Williams, L., Wiebe, E., Yang, K., Ferzli, M., & Miller, C. (2002). In support of pair programming in the introductory computer science course. *Computer Science Education*, *12*(3), 197-212.

Wilson, A.,& Moffat, D. C. (2010). Evaluating Scratch to introduce younger schoolchildren to programming. In *Proceedings of the 22nd Annual Workshop of the Psychology of Programming Interest Group* (pp. 63-75).

Wilson, B. C.,& Shrock, S. (2001). Contributing to success in an introductory computer science course: a study of twelve factors. *ACM SIGCSE Bulletin, 33(4),* 184-188.

Wilson, B. C. (2002). A study of factors promoting success in computer science including gender differences. *Computer Science Education*, *12*(1-2), 141-164.

Winslow, L. E. (1996). Programming pedagogy—a psychological overview. *ACM SIGCSE Bulletin*, *28*(3), 17-22.

Wray, S. (2007). SQ minus EQ can predict programming aptitude. In *Proceedings of the PPIG 19th Annual Workshop, Finland* (pp. 243-254).

Yurdugül, H., & Aşkar, P. (2013). Learning programming, problem solving and gender: A longitudinal study. *Procedia-Social and Behavioral Sciences*, *83*, 605-610.

Zaharija, G., Mladenović, S., & Boljat, I. (2013). Introducing basic Programming Concepts to Elementary School Children. *Procedia-Social and Behavioral Sciences*, *106*, 1576-1584.

Zualkernan, I. A., Allert, J., & Qadah, G. Z. (2006). Learning styles of computer programming students: a Middle Eastern and American comparison. *Education, IEEE Transactions on*, *49*(4), 443-450.

Zweben, S. (2011). Computing degree and enrollment trends. *Computing Research Association.*

EK 1: PROGRAMLAMA YETENEK TESTİ[1]

Adı - Soyadı : ……

Ölçekte yer alan sorulara verdiğiniz yanıtlar, kesinlikle size not vermek ya da sizi eleştirmek amacıyla kullanılmayacaktır. Ölçekte vereceğiniz cevaplarınız tamamıyla gizli tutulacak ve sadece araştırmacılar tarafından değerlendirilecektir; elde edilecek bilgiler bilimsel yayımlarda kullanılacaktır.

Bu sorular sizin programlamaya ne kadar yatkın olduğunuzu belirlemek içindir. Her sorunun bir doğru cevabı vardır.

1)	jkojaspfskasfk	jkojaspfskasfk
2)	iqtgiwegasdio	iqtgiyegasdio
3)	iqtgiwegasdio	iqtgiwegasdio
4)	wyesdsdlxvc	wyesdsdlxvc
5)	jasbhuhoqhwlk	jasbhuhoqhwlk
6)	poanksdnuqwb	poanksdnuqwb

1. Yukarıdaki 6 satırda sol ve sağ taraf aynı karakter kümelerinden oluşmuştur, fakat bir satır farklıdır. Bu farklı satır hangisidir?

A) 1 **B)** 2 **C)** 3 **D)** 4 **E)** 5 **F)** 6

1)	f7y349hwe[nfl';SD,|M	f7y349hwe[nfl';SD,|M
2)	CDBI98QEy39o;@:lIP	CDBI98QEy39o;@:lIP
3)	\lvnrllrvoprr~f.4'wr	\lvnrllrvoprr~f.4'wr
4)	;sapr){3I4JM\@~mm	;sapr){3I4JM\@~mm
5)	dsNvn(Pos\o;l\md;)	dsNvn(Pos\e;l\md;)
6)	pUJ304'f.:gWV,[c\;IMq;I	pUJ304'f.:gWV,[c\;IMq;I

2. Yukarıdaki 6 satırda sol ve sağ taraf aynı karakter kümelerinden oluşmuştur, fakat bir satır farklıdır. Bu farklı satır hangisidir?

A) 1 **B)** 2 **C)** 3 **D)** 4 **E)** 5 **F)** 6

[1] Kullanım izni Ek-6'da verilmiştir. Kent Üniversitesi Kariyer Danışma Servisi'nin izni ile üretilmiştir www.kent.ac.uk/careers. Detaylar için Bruce Woodcock ile iletişime geçiniz bw@kent.ac.uk.

1)	'z;agle5pao5[,zd,pE;"	'z;agle5pao5[,zd,pE;"
2)	\kjfo5;gm"va'a'c,'\fz	\kjfo5;gm"va'a'c,'\fz
3)	zk;rgkggr'lz#g#[[pr[s5o;	zk;rgkggr'lz#g#[[pr[s5o;
4)	[aepv5[.[3.q3#[kca;fh'	[aepv5[.[3.q3#[kca;fh'
5)	oaejo[v,5./Av?cva;VS'	oaejo[v,5./Av?cva;VS'
6)	ladnl:vdlne,842rybqi\m	ladnl:vdlne;842rybqi\m

3. Yukarıdaki 6 satırda sol ve sağ taraf aynı karakter kümelerinden oluşmuştur, fakat bir satır farklıdır. Bu farklı satır hangisidir?

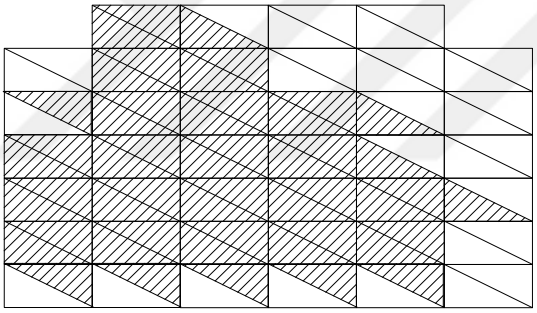
A) 1 **B)** 2 **C)** 3 **D)** 4 **E)** 5 **F)** 6

4. Ahmet aklından bir sayı tutuyor. Bu sayının karesini alıyor, 5 çıkartıyor, 4 ile çarpıyor, 7 çıkartıyor 3'e bölüyor ve son olarak 6 ekliyor. Sonuç 9 ise, Ahmet'in aklından tuttuğu sayı kaçtır?

A) 1 **B)** 2 **C)** 3 **D)** 4 **E)** 5 **F)** 6

5. Bir saatin akrebi öğleden sonra 2'den öğleden önce 9'a geri alınırsa, akrep kaç derece dönmüş olur?

A) 120 **B)** 135 **C)** 150 **D)** 165 **E)** 180 **F)** 205



6. Yukarıdaki şekilde taranmış alan, toplam alanın yaklaşık yüzde kaçıdır?

A) 60 **B)** 63 **C)** 66 **D)** 69 **E)** 72 **F)** 75

7, 8 ve 9. soruları aşağıdaki alfabeye göre çözün

ALFABE: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

7. Eğer ADD=9, BAD=7 ve CAD=8 ise ADA kaçtır?

A) 3 **B)** 4 **C)** 5 **D)** 6 **E)** 7 **F)** 8

8. Eğer BAD=10, DAC=11, ve CGI=22 ise, OCCAM kaçtır?

A) 35 **B)** 36 **C)** 37 **D)** 39 **E)** 40 **F)** Hiçbiri

9. Eğer DATA=52, CACHE=40, ve BIT=62 ise BABBAGE kaçtır?

A) 40 **B)** 41 **C)** 42 **D)** 43 **E)** 44 **F)** Hiçbiri

10. Yönünüz kuzeye doğrudur. Sola 90 derece dönün. 180 derece sağa dönün. Yönünüzü ters çevirin. Sola 45 derce dönün. Yönünüzü ters çevirin. Sağa 270 derce dönün. Şimdi yönünüz nedir?

A) Kuzey **B)** Batı **C)** Güney Doğu

D) Güney Batı **E)** Kuzey Batı **F)** Hiçbiri

Bir uçak firması biletleri şu şekilde kodlamaktadır:

- İngiltere'ye uçuşlar A, Avrupa'ya uçuşlar B, Asya'ya uçuşlar C, ve Amerika'ya uçuşlar D ile kodlanmaktadır.
- Eğer uçuşlar akşam 10 ile sabah 6 arasında ise, aynı harflerin küçükleri kullanılmaktadır (a,b,c, ve d)
- Bay yolcular X, bayan yolcular Y ile kodlanmaktadır. Çocuklar için küçük harfler kullanılmaktadır (x ve y)
- Yemekler şu şekilde kodlanmaktadır: Avrupa yemekleri G, Asya yemekleri H, Vejeteryen (etyemez) K. Çocuklar için aynı harflerin küçükleri kullanılmaktadır. (g,h,k)
- Birinci sınıf yolcular P, iş adamları Q, Ekonomi sınıfı R ile kodlanmaktadır

11. Ekonomi sınıfında sabah 5 uçağı ile Paris'e giden 8 yaşındaki vejeteryen (etyemez) kız çocuk için kod nedir?

A) bYkR **B)** bykr **C)** bykR **D)** BykP **E)** aykR **F)** Hiçbiri

12. Üç bilgisayar bir sıra halinde dizilmektedir. Dell (D), Exper (E)'in solunda fakat yanında olmak zorunda değildir. Mavi bilgisayar Beyaz bilgisayarın sağındadır. Siyah bilgisayar Casper (C)'in solundadır. Casper, Exper'in solundadır. Bilgisayarların soldan sağa sıralanışı nasıldır?

A) E, C, D **B)** E, D, C **C)** C, D, E **D)** C, E, D **E)** D, E, C **F)** D, C, E

13. Tuncay, verilen bir kutu şekerin üçte birini yeyince, kendisini hasta hissediyor ve duruyor. Sonraki gün kalanın üçte birini yiyor. Bir sonraki gün ise kalanın üçte birini yiyor. Geriye 8 şeker kalmış ise, Tuncay'a kaç şeker verilmiştir?

A) 18 **B)** 21 **C)** 24 **D)** 27 **E)** 30 **F)** 33

14 ve 15. soruları aşağıdaki bilgilere göre çözün

Akıllı maymunların kullandığı bir sayı sistemi şu şekildedir:

- Bir muz = 1
- Bir portakal ve iki muz = 6
- Bir portakal yarım mango değerindedir

14. İki mango, bir portakal ve bir muzun sayı değeri kaçtır?

A) 21 **B)** 24 **C)** 27 **D)** 30 **E)** 33 **F)** 36

15. İki mango ve bir portakalın bir portakal ve bir muza bölümünün meyve cinsinden değeri nedir?

A) Bir mango **B)** Bir muz **C)** Bir portakal

D) İki muz **E)** Üç muz **F)** Bir portakal ve bir muz

16, 17 ve 18. soruları aşağıdaki alfabeye göre çözün:

ALFABE: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

16. Eğer JAVA'nın kodu LCXC ise, BASIC'in kodu nedir?

A) CBTJD **B)** DCUKE **C)** EDVLF **D)** FEWMG **E)** CDFFG **F)** Hiçbiri

17. Eğer FORTRAN'ın kodu GMUPWUU ise PASCAL'ın kodu nedir?

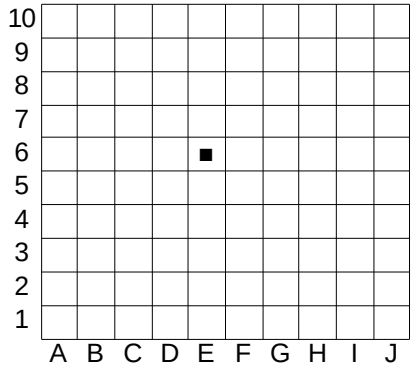
A) QYVFG **B)** QCVGFR **C)** QCPGVR

D) GMPGFR **E)** QCVXFF **F)** Hiçbiri

18. Eğer PHP'nin kodu QLY ise. SQL'in kodu nedir?

A) TUU **B)** TUS **C)** TRM **D)** TUB **E)** VUS **F)** Hiçbiri

19 ve 20. soruları aşağıdaki şekle göre cevaplayınız.



19. 6E kutusunda Doğu'ya doğru bakmaktasınız. 3 kutu ilerleyin. Saat yönünde 90 derece dönün, iki kutu ilerleyin, saat yönünün tersinde 180 derece dönün. 5 kutu ilerleyin, saat yönünün tersine 90 derece dönün. 4 kutu ilerleyin, saat yönünde 90 derece dönün. İki kutu geri gidin. Şu anda bulunduğunuz konumun Y koordinatı nedir?

A) 6 **B)** 7 **C)** 8 **D)** 9 **E)** 10 **F)** 11

20. 6E kutusunda Güney-Batı'ya doğru bakmaktasınız. Üç kutu ilerleyin. Saat yönünde 135 derece dönün. 4 kutu ilerleyin. Saat yönünde 45 derece dönün. 2 kutu ilerleyin. Saat yönünün tersine 90 derece dönün ve 4 kutu ilerleyin. Bulunduğunuz konumun X koordinatı nedir?

A) C **B)** D **C)** E **D)** F **E)** G **F)** H

21. Saat 10:30'da akrep ve yelkovan arasındaki açı kaç derecedir?

A) 75 **B)** 90 **C)** 105 **D)** 120 **E)** 135 **F)** 150

Bir çikolata fabrikasında, bir makine 1 kg'lık çikolata kalıplarını alıp 10gr'lık diktörtgenler halinde kesmektedir. Daha sonra bu diktörtgen çikolatalar, 6gr lık silindirler haline getiriliyor. Artan çikolatalar boşa gitmektedir. Silindir halindeki çikolatalardan 4 tanesi bir poşete konulmakta ve bu poşetlerden 6 tanesi bir kutuya konularak paktetlenmektedir.

22. Bu kutulardan tam dolu olanları, orjinal kilogramın yüzde kaçına denk gelir?

A) %58 **B)** %60 **C)** %62 **D)** %64 **E)** %66 **F)** %68

23, 24 ve 25. soruları aşağıdaki şekle göre cevaplayınız. Burada bir hücrenin koordinatları o hücrenin değerini verir.

	A	B	C	D	E	F
1	9	7	8	4	0	0
2	8	2	3	7	0	0
3	11	1	5	6	0	0
4	13	9	6	3	0	0

23. A4, D3 ile çarpılıp C2'ye bölünürse sonuç kaç olur?

A) 24 **B)** 26 **C)** 28 **D)** 30 **E)** 33 **F)** Hiçbiri

24. B4 artı A2 sonucunu F1'de saklayın. A4 eksi D2 sonucunu F3'de saklayın. F1 ile F3 çarpımı kaç olur?

A) 17 **B)** 84 **C)** 96 **D)** 104 **E)** 108 **F)** Hiçbiri

Basamak 1: C3 ile D4'ü çarpın sonucu F4'de saklayın

Basamak 2: F4'ü 3 ile çarpın ve sonucu F4'de saklayın. E3'e 1 ekleyin

Basamak 3: E3'ün değeri 3 olana kadar Basamak 2'yi tekrarlayın

25. F4'ün değeri nedir?

A) 45 **B)** 345 **C)** 405 **D)** 450 **E)** 1215 **F)** Hiçbiri

26. A1+B3+C4+D2 sonucunu E2'de saklayın. A3+B1+C2+D4 sonucunu E4'de saklayın. Eğer E4, E2'den büyükse, değerlerini değiş tokuş (swap) yapın, değilse olduğu gibi bırakın. E2 ile D1'i çarpıp sonuçtan A4'ü çıkarın ve sonucu F2'de saklayın. F2'nin değeri kaçtır.

A) 79 **B)** 83 **C)** 96 **D)** 95 **E)** 94 **F)** Hiçbiri

EK 2: PROGRAMLAMA BAŞARI TESTİ

T.C. Ondokuz Mayıs Üniversitesi / Eğitim Fakültesi

BÖTE – Programlama Dilleri I - Ara Sınavı

Yönerge:Aşağıda aynı sorunun sol tarafta Python sağ tarafta Blockly kodları vardır.Cevaplarınızı sondaki CEVAP KAĞIDI'na işaretleyiniz. ***CEVAP KAĞIDI****'na* işaretlenmeyen cevaplar geçersizdir. Her doğru cevap 4 puandır. Yanlış doğruyu götürmez. *SINAV SÜRESİ 40 dakikadır.*

1. Aşağıdaki programın çıktısı nedir?

```
a = 3
b = '4'
print(a * b)
```

A) 12

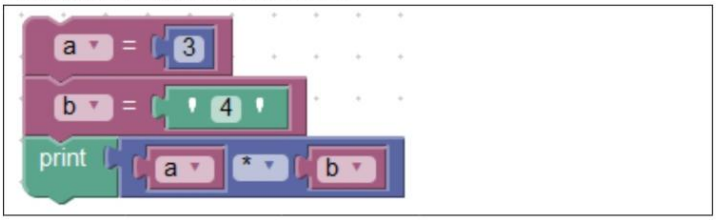
B) 3333

C) 444

D) 4444

E) 333

1. Aşağıdaki programın çıktısı nedir?



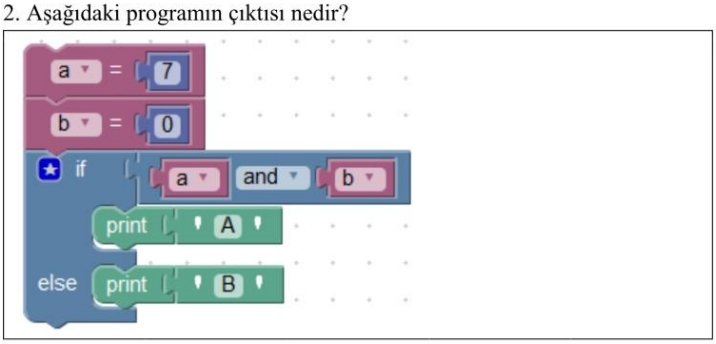
A) 12 B) 3333 C) 444 D) 4444 E) 333

2. Aşağıdaki programın çıktısı nedir?

```
a = 7
b = 0
if (a and b):
  print('A')
else:
  print('B')
```

A) A B) B C) A B D) B A E) A A

2. Aşağıdaki programın çıktısı nedir?



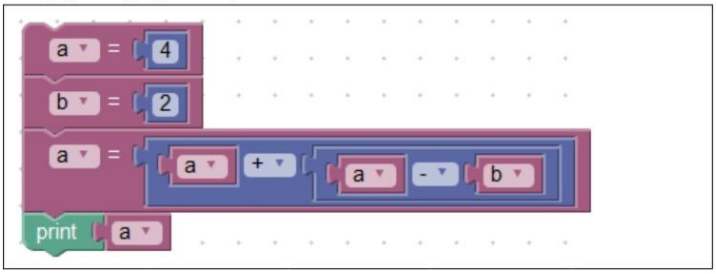
A) A B) B C) A B D) B A E) A A

3. Aşağıdaki programın çıktısı nedir?

```
a = 4
b = 2
a = a + a - b
print(a)
```

A) 4 B) 2 C) 8 D) 0 E) 6

3. Aşağıdaki programın çıktısı nedir?



A) 4 B) 2 C) 8 D) 0 E) 6

4. Aşağıdaki programın çıktısı nedir?

```
x = 4
y = x * str(3)
print(y)
```

A) 12

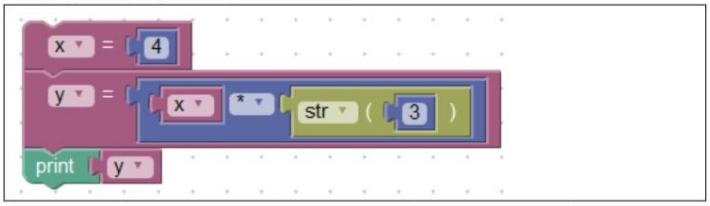
B) 3333

C) 444

D) 4444

E) 333

4. Aşağıdaki programın çıktısı nedir?



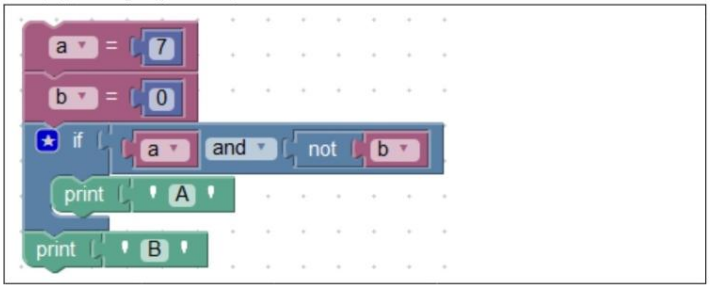
A) 12 B) 3333 C) 444 D) 4444 E) 333

5. Aşağıdaki programın çıktısı nedir?

```
a = 7
b = 0
if (a and not b):
  print('A')
print('B')
```

A) A B) B C) A B D) B A E) A A

5. Aşağıdaki programın çıktısı nedir?



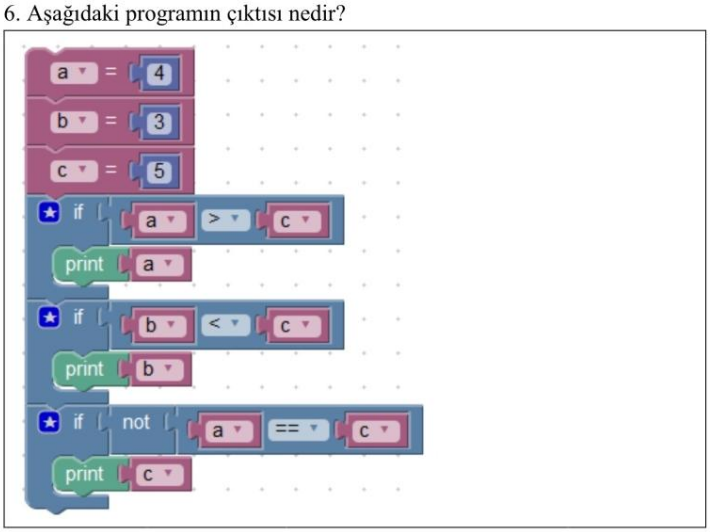
A) A B) B C) A B D) B A E) A A

6. Aşağıdaki programın çıktısı nedir?

```
a = 4
b = 3
c = 5
if (a > c):
  print(a)
if (b < c):
  print(b)
if (not (a == c)):
  print(c)
```

A) 5 3 B) 3 5 C) 4 3 5 D) 4 3 E) 3 4

6. Aşağıdaki programın çıktısı nedir?



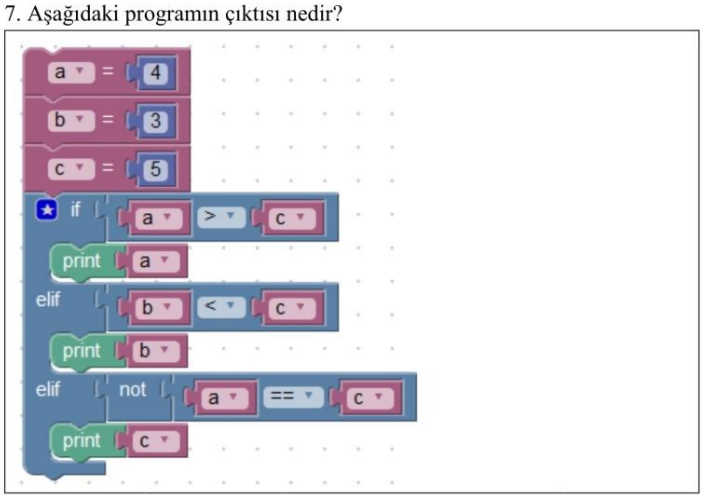
A) 5 3 B) 3 5 C) 4 3 5 D) 4 3 E) 3 4

7. Aşağıdaki programın çıktısı nedir?

```
a = 4
b = 3
c = 5
if (a > c):
  print(a)
elif (b < c):
  print(b)
elif (not (a == c)):
  print(c)
```

A) 3 B) 4 C) 5 D) 4
3 E) 3
5

7. Aşağıdaki programın çıktısı nedir?



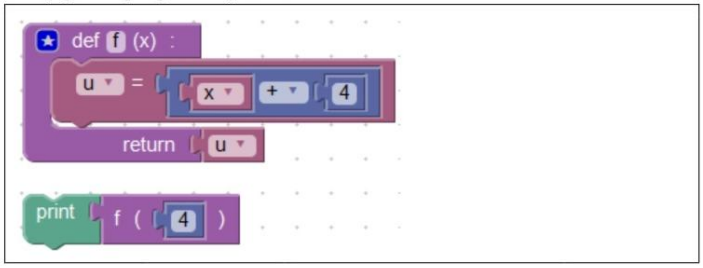
A) 3 B) 4 C) 5 D) 4
3 E) 3
5

8. Aşağıdaki programın çıktısı nedir?

```
def f(x):
  u = x + 4
  return u

print(f(4))
```

A) 4 B) 2 C) 8 D) 0 E) 6

8. Aşağıdaki programın çıktısı nedir?



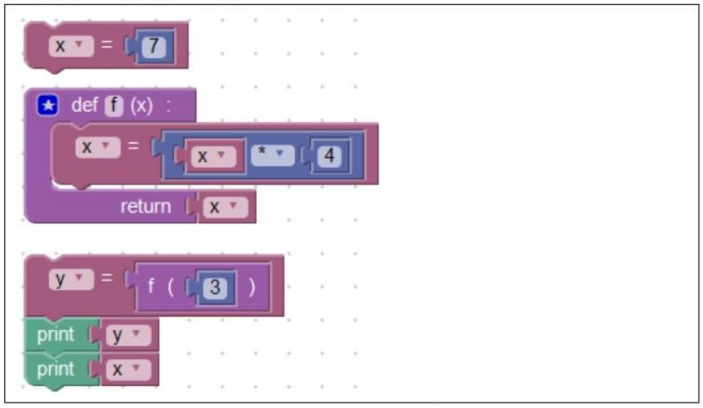
A) 4 B) 2 C) 8 D) 0 E) 6

9. Aşağıdaki programın çıktısı nedir?

```
x =7
def f(x):
  x = x *4
  return x

y = f(3)
print(y)
print(x)
```

A) 12
4 B) 12
7 C) 7
12 D) 7
4 E) 4
7

9. Aşağıdaki programın çıktısı nedir?



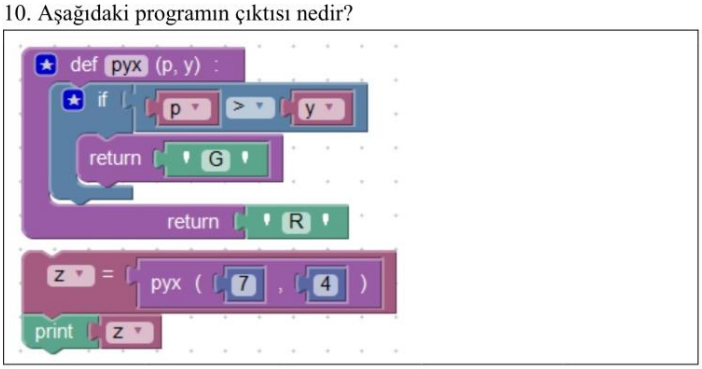
A) 12
4 B) 12
7 C) 7
12 D) 7
4 E) 4
7

10. Aşağıdaki programın çıktısı nedir?

```
def pyx(p, y):
  if (p > y):
    return ('G')
  return 'R'

z = pyx(7, 4)
print(z)
```

A) G B) R C) R D) G E) R
R G R

10. Aşağıdaki programın çıktısı nedir?



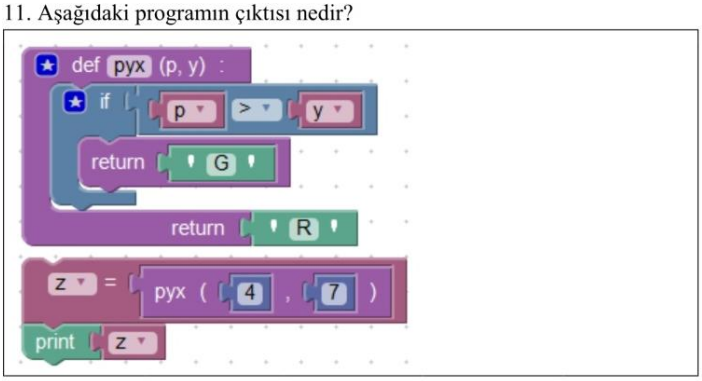
A) G B) R C) R D) G E) R
R G R

11. Aşağıdaki programın çıktısı nedir?

```
def pyx(p, y):
  if (p > y):
    return ('G')
  return 'R'

z = pyx(4, 7)
print(z)
```

A) G B) R C)G D)R E) R
R G R

11. Aşağıdaki programın çıktısı nedir?



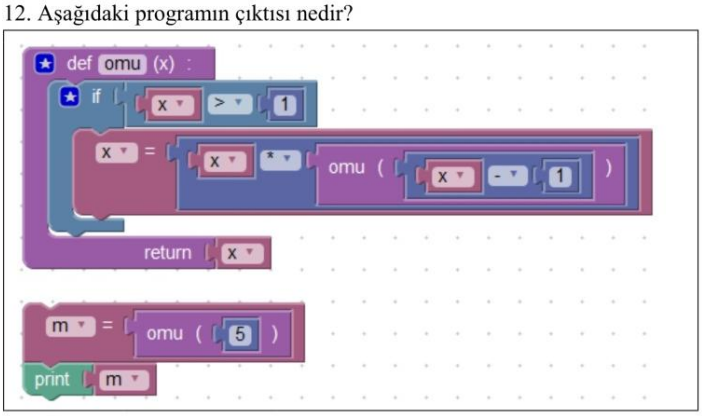
A) G B) R C)G D)R E) R
R G R

12. Aşağıdaki programın çıktısı nedir?

```
def omu(x):
  if (x > 1):
    x = x * omu(x - 1)
  return x

m = omu(5)
print(m)
```

A) 4 B) 100 C) 105 D) 120 E) 5

12. Aşağıdaki programın çıktısı nedir?



A) 4 B) 100 C) 105 D) 120 E) 5

13. Aşağıdaki programın çıktısı nedir?

```
L = [1, 9, 14, 8, 6]
print(L[2] - L[-2])
```

A) 6 B) 8 C) 4 D) 12 E) 0

13. Aşağıdaki programın çıktısı nedir?

A) 6 B) 8 C) 4 D) 12 E) 0

14. Aşağıdaki programın çıktısı nedir?

```
i = 'M'
s = 'SAMSUN'
L = []
for i in s:
  L.append(s.index(i))
print(i)
```

A) S

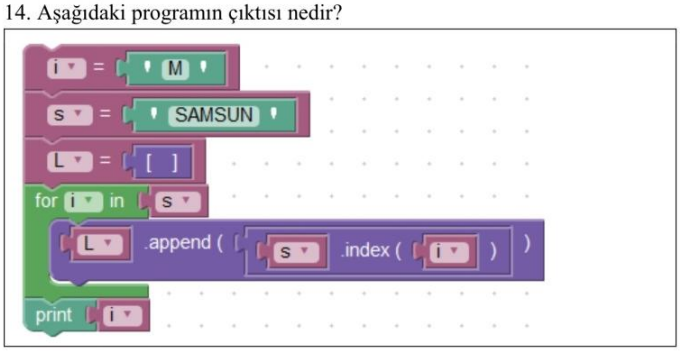
B) M

C) N

D) Hata verir

E) None

14. Aşağıdaki programın çıktısı nedir?



A) S

B) M

C) N

D) Hata verir

E) None

15. Aşağıdaki programın çıktısı nedir?

```
L = [1, 9, 14, 4, 6]
L.append(8)
L.remove(9)
L.pop(4)
L.reverse()
print(L)
```

A) [6, 4, 1]

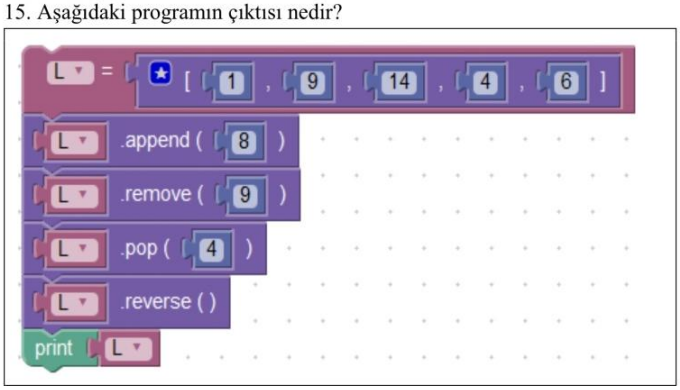
B) [6, 4, 1, 9]

C) [6, 4, 1, 8]

D) [6, 4, 1, 9, 8]

E) [6, 4, 14, 1]

15. Aşağıdaki programın çıktısı nedir?



A) [6, 4, 1]

B) [6, 4, 1, 9]

C) [6, 4, 1, 8]

D) [6, 4, 1, 9, 8]

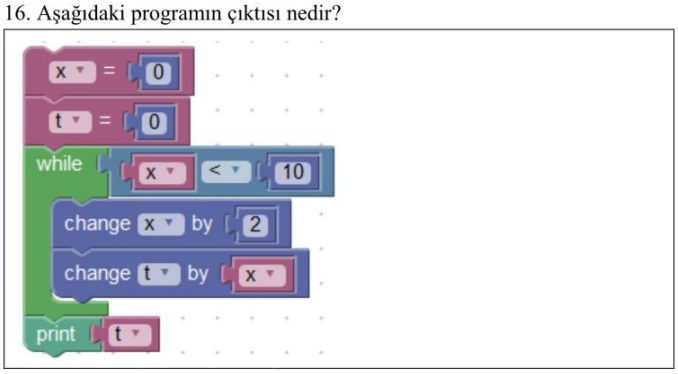
E) [6, 4, 14, 1]

16. Aşağıdaki programın çıktısı nedir?

```
x = 0
t = 0
while (x < 10):
  x += 2
  t += x
print(t)
```

A) 10 B) 20 C) 30 D) 12 E)18

16. Aşağıdaki programın çıktısı nedir?



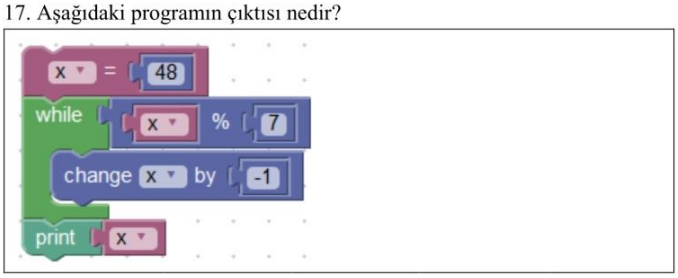
A) 10 B) 20 C) 30 D) 12 E) 18

17. Aşağıdaki programın çıktısı nedir?

```
x = 48
while (x % 7):
  x -= 1
print(x)
```

A) 6.85 B) 45 C) 44 D) 43 E) 42

17. Aşağıdaki programın çıktısı nedir?



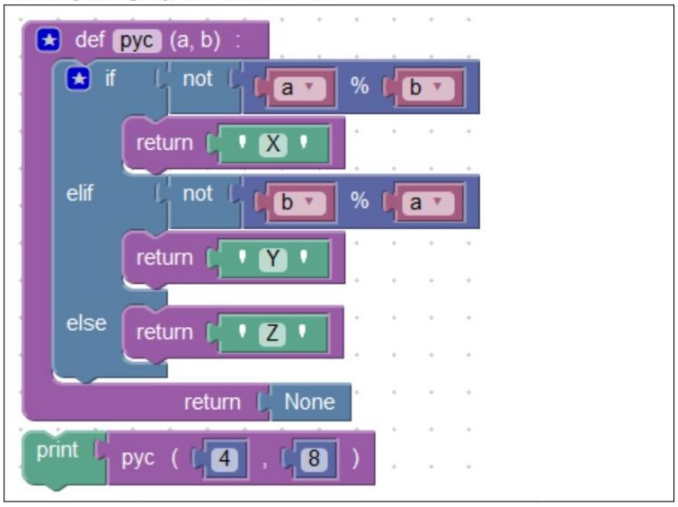
A) 6.85 B) 45 C) 44 D) 43 E) 42

18. Aşağıdaki programın çıktısı nedir?

```
def pyc(a, b):
  if (not (a % b)):
    return ('X')
  elif (not (b % a)):
    return ('Y')
  else:
    return ('Z')
  return None

print(pyc(4, 8))
```

A) X B) Y C) Z D) Y Z E) X Z

18. Aşağıdaki programın çıktısı nedir?



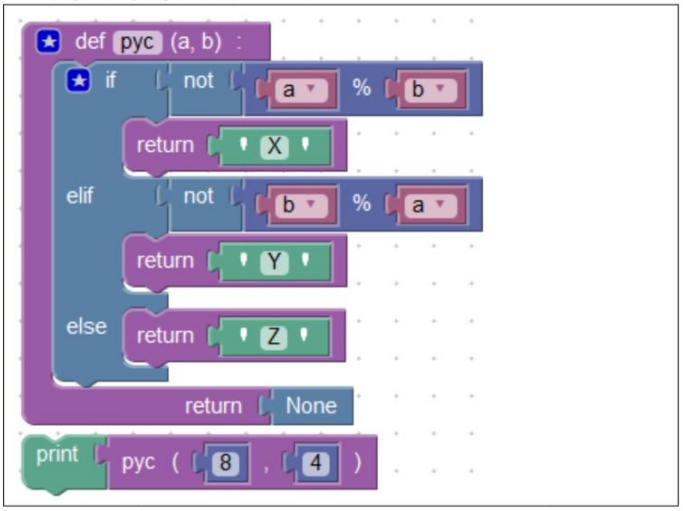
A) X B) Y C) Z D) Y Z E) X Z

19. Aşağıdaki programın çıktısı nedir?

```
def pyc(a, b):
  if (not (a % b)):
    return ('X')
  elif (not (b % a)):
    return ('Y')
  else:
    return ('Z')
  return None

print(pyc(8, 4))
```

A) X B) Y C) Z D) Y Z E) X Z

19. Aşağıdaki programın çıktısı nedir?



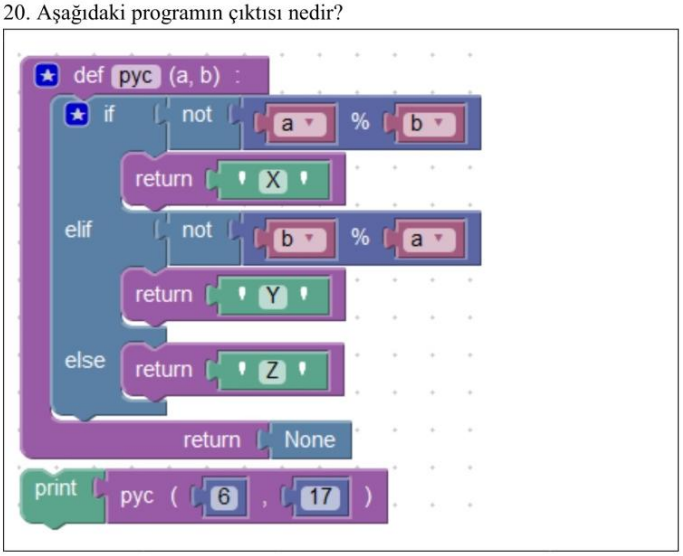
A) X B) Y C) Z D) Y Z E) X Z

20. Aşağıdaki programın çıktısı nedir?

```
def pyc(a, b):
  if (not (a % b)):
    return ('X')
  elif (not (b % a)):
    return ('Y')
  else:
    return ('Z')
  return None

print(pyc(6, 17))
```

A) Y Z B) X Z C) X D) Y E) Z

20. Aşağıdaki programın çıktısı nedir?



A) Y Z B) X Z C) X D) Y E) Z

21. Aşağıdaki programın çıktısı nedir?

```
L = [1, 9, 14, 4, 6]
S = []
for u in L:
  if (u % 3):
    S.append(u)
print(S)
```

A) [1, 14, 4]

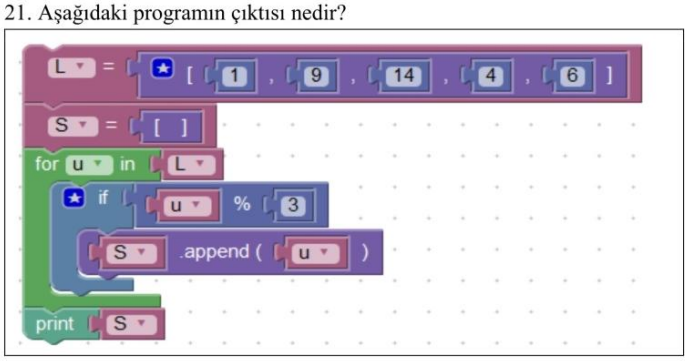
B) [9, 6]

C) [1, 0, 2, 1, 0]

D) [1, 9, 14]

E) [4, 6]

21. Aşağıdaki programın çıktısı nedir?



A) [1, 14, 4]

B) [9, 6]

C) [1, 0, 2, 1, 0]

D) [1, 9, 14]

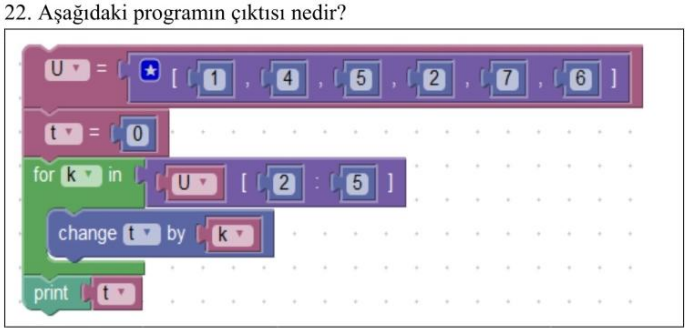
E) [4, 6]

22. Aşağıdaki programın çıktısı nedir?

```
U = [1, 4, 5, 2, 7, 6]
t = 0
for k in U[2:5]:
  t += k
print(t)
```

A) 17 B) 16 C) 15 D) 14 E) 13

22. Aşağıdaki programın çıktısı nedir?



A) 17 B) 16 C) 15 D) 14 E) 13

23. Aşağıdaki programın çıktısı nedir?

```
K = [1, 8, 1, 6, 9, 8, 6, 6]
U = []
for m in K:
  if (not m in U):
    U.append(m)
print(U)
```

A) [1, 8, 6]

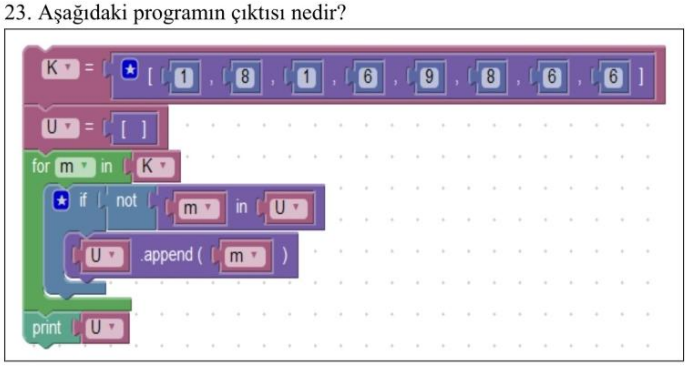
B) [9]

C) [1, 8, 6, 9]

D) [1, 8, 1, 6, 9, 8, 6]

E) [6]

23. Aşağıdaki programın çıktısı nedir?



A) [1, 8, 6] B) [9] C) [1, 8, 6, 9] D) [1, 8, 1, 6, 9, 8, 6] E) [6]

24. Aşağıdaki programın çıktısı nedir?

```
X = [9, 12, 4, 5, 7, 9]
Y = [5, 9, 5, 7]
M = []
for i in X:
  if (not i in M):
     M.append(i)
for i in Y:
  if (not i in Y):
     M.append(i)
print(M)
```

A) [5, 9, 5, 7]

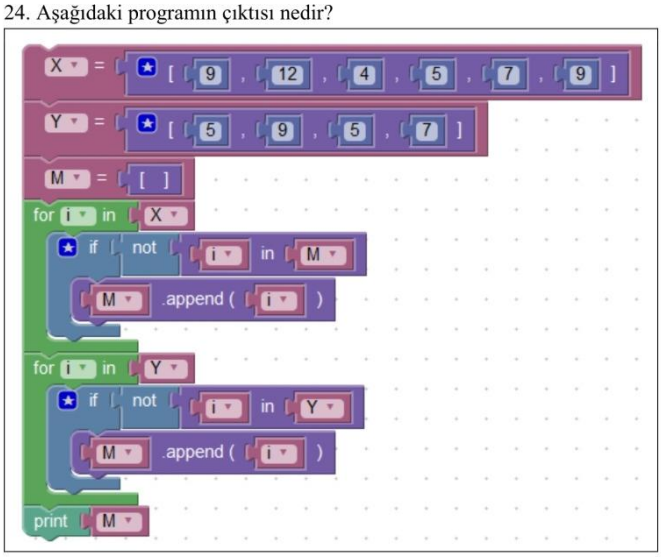
B) [9, 12, 4, 5, 7, 9, 5, 9, 5, 7]

C) [9, 12, 4, 5, 7]

D) [5, 9, 7]

E) [12, 4]

24. Aşağıdaki programın çıktısı nedir?



A) [5, 9, 5, 7]

B) [9, 12, 4, 5, 7, 9, 5, 9, 5, 7]

C) [9, 12, 4, 5, 7]

D) [5, 9, 7]

E) [12, 4]

25. Aşağıdaki programın çıktısı nedir?

```
X = [12, 4, 3, 12, 8]
Y = [8, 4, 5, 12, 4]
M = []
for i in X + Y:
  if (not i in M):
     M.append(i)
print(M)
```

A) [12, 4, 3, 12, 8, 8, 4, 5, 12]

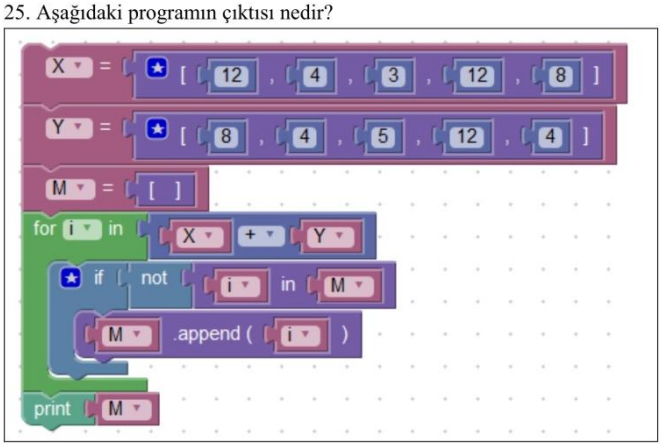
B)[12, 4, 8]

C) [3, 5]

D) [12, 4]

E) [12, 4, 3, 8, 5]

25. Aşağıdaki programın çıktısı nedir?



A) [12, 4, 3, 12, 8, 8, 4, 5, 12]

B) [12, 4, 8]

C) [3, 5]

D) [12, 4]

E) [12, 4, 3, 8, 5]

EK 3: BİLGİSAYAR PROGRAMLAMAYA KARŞI TUTUM ÖLÇEĞİ[2]

Adı-Soyadı : ……..

Ölçekte yer alan sorulara verdiğiniz yanıtlar, kesinlikle size not vermek ya da sizi eleştirmek amacıyla kullanılmayacaktır. Bu soruların herkes için geçerli doğru yanıtları bulunmamaktadır. Bu nedenle lütfen aşağıda verilen tüm soruları dikkatle okuyarak cevabınızı, ifadenin karşısındaki seçeneklerden sizin için en uygun olanı işaretleyerek belirtiniz. Verilen ifadeye "Kesinlikle katılmıyorsanız" 1'i, "Katılmıyorsanız" 2'yi, "Kısmen katılıyorsanız" 3'ü, "Katılıyorsanız" 4'ü, "Kesinlikle katılıyorsanız" 5'i işaretleyiniz. Ölçekte vereceğiniz cevaplarınız tamamıyla gizli tutulacak ve sadece araştırmacılar tarafından değerlendirilecektir; elde edilecek bilgiler bilimsel yayımlarda kullanılacaktır.

		Kesinlikle Katılmıyorum	**Katılmıyorum**	**Kararsızım**	**Katılıyorum**	**Kesinlikle Katılıyorum**
S1	Bilgisayar programlama problemlerini çözmede kendime güveniyorum.	1	2	3	4	5
S2	Programlamayı öğreneceğimden eminim.	1	2	3	4	5
S3	Daha zor programlama problemlerinin üstesinden geleceğimi düşünüyorum.	1	2	3	4	5
S4	Programlama derslerinde iyi notlar alabilirim.	1	2	3	4	5
S5	Söz konusu programlama olunca kendime güvenim çoktur.	1	2	3	4	5
S6	Programlamada iyi değilim.	1	2	3	4	5
S7	İleri programlama derslerinde başarılı olacağımı sanmıyorum.	1	2	3	4	5

[2] Başer (2013a)'dan alınmıştır.

S8	Bilgisayar programlamayı iyi yapan birisi değilim.	1	2	3	4	5
S9	Çok çalışmama rağmen programlama bana inanılmaz zor geliyor.	1	2	3	4	5
S10	Birçok dersin üstesinden geliyorum fakat programlama problemlerinde becerim yok.	1	2	3	4	5
S11	En başarısız olduğum ders bilgisayar programlamadır.	1	2	3	4	5
S12	Programlama derslerinde mükemmel öğrenci olarak algılanmak beni mutlu eder.	1	2	3	4	5
S13	Programlama derslerinde göze çarpan bir öğrenci olmak bana gurur verir.	1	2	3	4	5
S14	Programlama derslerinde en yüksek notları almak beni mutlu eder.	1	2	3	4	5
S15	Programlama yarışmalarında bir ödül almak benim için muhteşemdir.	1	2	3	4	5
S16	Programlama yarışmasında birinci olmak beni sevindirir.	1	2	3	4	5
S17	Programlama derslerinde akıllı bir öğrenci sayılmak benim için büyük bir şeydir.	1	2	3	4	5
S18	Programlama ile ilgili bir yarışmada bir ödül kazanmak bana cazip gelmez.	1	2	3	4	5
S19	Programlama derslerinde yüksek not alırsam onu önemsemem.	1	2	3	4	5
S20	Eğer programlama dersinde bir kere yüksek not alırsam bir daha istemem.	1	2	3	4	5
S21	Programlama derslerinde başarılı bir öğrenci olursam insanlar beni daha az sever.	1	2	3	4	5
S22	İnsanların programlama derslerinde akıllı öğrenci olduğumu düşünmelerini istemem.	1	2	3	4	5
S23	Sonraki çalışmalarımda programlamaya ihtiyacım olacak.	1	2	3	4	5
S24	Programlamaya çalışıyorum çünkü onun ne kadar gerekli olduğunu biliyorum.	1	2	3	4	5
S25	Programlamayı bilmek iş bulmama yardımcı olacak.	1	2	3	4	5

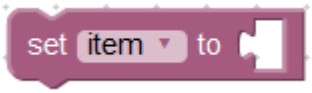
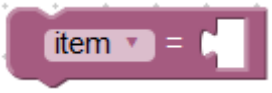
S26	Programlama değerli ve yararlıdır.	1	2	3	4	5
S27	İş hayatımda programlamayı birçok şekilde kullanacağım.	1	2	3	4	5
S28	İş hayatım için programlamanın önemi yok.	1	2	3	4	5
S29	İş hayatımda programlamanın önemi olmayacak.	1	2	3	4	5
S30	İş hayatımda programlamayı seyrek kullanacağım.	1	2	3	4	5
S31	Programlama dersleri almak zaman kaybıdır.	1	2	3	4	5
S32	Okul sonrasında programlamanın çok az kullanım yeri olacağını tahmin ediyorum.	1	2	3	4	5
S33	Bir programlama problemi ile karşılaşırsam onu çözünceye kadar uğraşırım.	1	2	3	4	5
S34	Bir program üzerinde çalışmaya başlarsam durmakta zorlanırım.	1	2	3	4	5
S35	Programlama dersinde cevapsız bir soru kalırsa, daha sonra hakkında düşünmeye devam ederim.	1	2	3	4	5
S36	Hemen anlayamadığım bilgisayar programlama problemleri beni kamçılar.	1	2	3	4	5
S37	Programlama probleminin zor olması ilgimi çekmez.	1	2	3	4	5
S38	İnsanların program yazarken çok zaman harcamaları ve bundan zevk almalarını anlamıyorum.	1	2	3	4	5

EK 4: BLOCKLY YAZILIMINDA YAPILAN DEĞİŞİKLİKLER

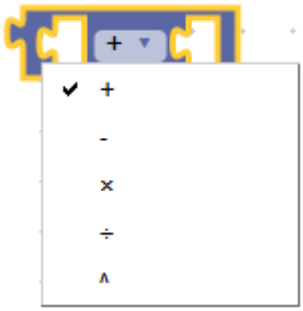
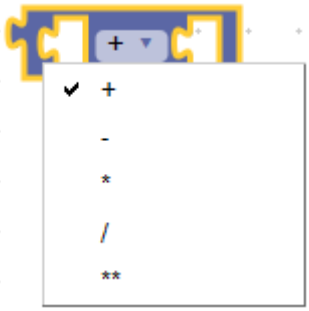
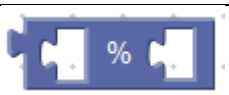
A-) Program Blokları ile İlgili Yapılan Değişiklikler

Bloklar ile ilgili yapılan değişiklikler eski ve yeni halleri karşılaştırılarak verilmiş ve gerekli açıklamalar yapılmıştır.

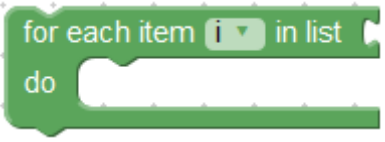
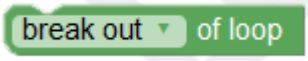
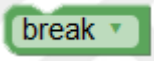
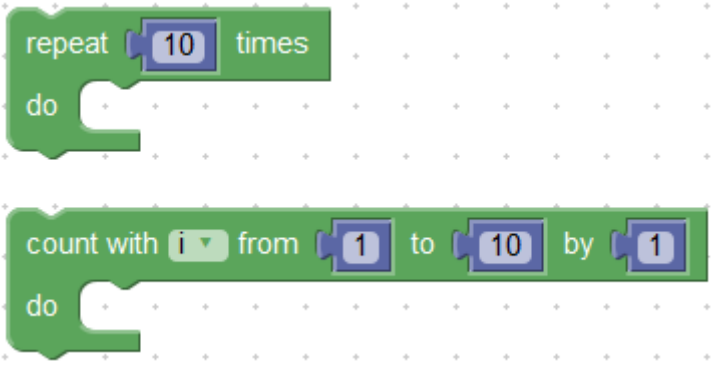
A.1. Değişken Bloğu

Eski	**Yeni**
	
Açıklama: Python programlama diline benzemesi açısından blokta yer alan *"set"* ve *"to"* kısımları kaldırılmış ve yerine *"="* işareti konulmuştur.	

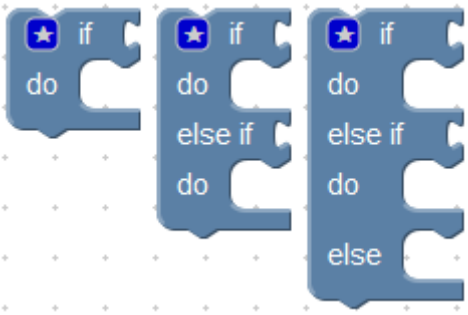
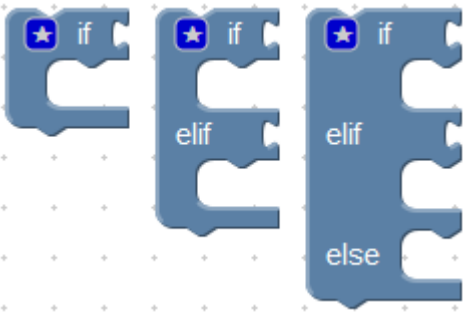
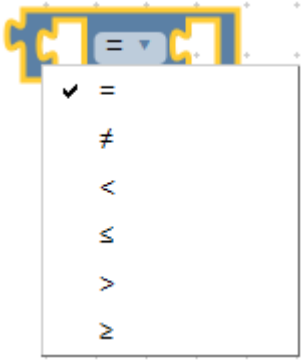
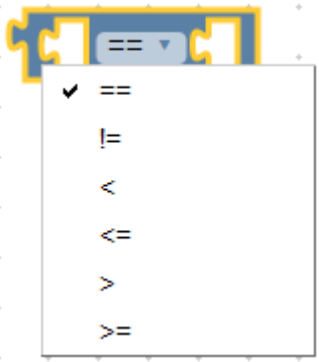
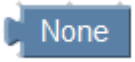
A.2. İşleç Blokları

Eski	**Yeni**
	
Açıklama: Çarpma, bölme ve üs alma işleçlerine ait işaretler Python diline göre değiştirilmiştir.	
	
Açıklama: İki sayının bölümünden kalanı bulmak için kullanılan blok Python'a uygun bir şekilde değiştirilmiştir.	

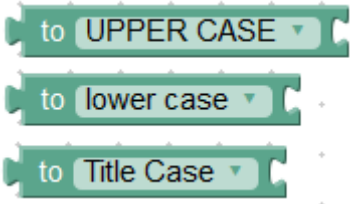
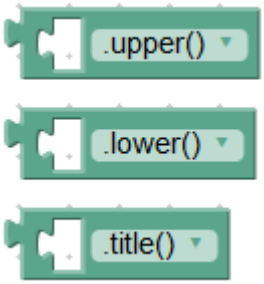
A.3. Döngü Blokları

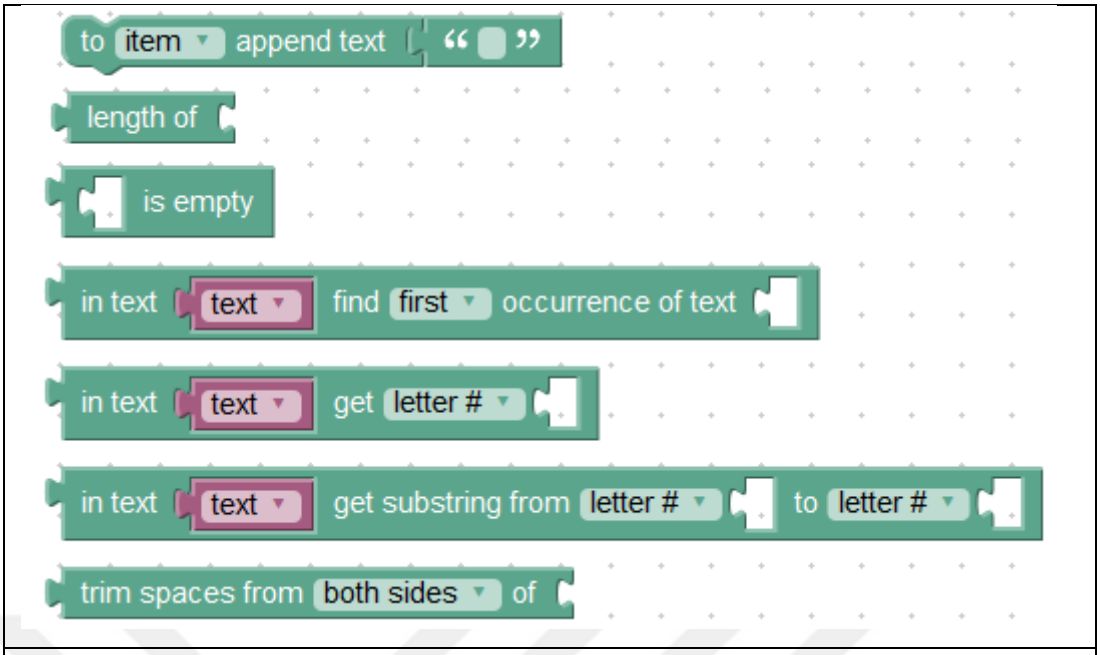
<table>
<tr><th>Eski</th><th>Yeni</th></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">Açıklama: While döngüsüne ait blok Python sözdizimine göre değiştirilerek daha sade bir hale getirilmiştir.</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">Açıklama: Bir liste üzeride iterasyon yapmak için kullanılan blok Python diline uygun bir şekilde değiştirilmiştir.</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">Açıklama: Döngüden çıkmak için kullanılan blok Python diline uygun bir şekilde değiştirilmiştir.</td></tr>
<tr><td colspan="2"> 
</td></tr>
<tr><td colspan="2">Açıklama: Yukarıdaki blokların çalışmada kullanılması gerekli olmadığı için ve yazılımın daha sade olması için çıkarılmasına karar verilmiştir.</td></tr>
</table>

A.4. Karşılaştırma ve Mantıksal İfade Blokları

<table>
<tr><th>Eski</th><th>Yeni</th></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">Açıklama: If karşılaştırma bloğu sözdizimi Python'a uygun olması için şekildeki gibi değiştirilmiştir.</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">Açıklama: True ve False ifadelerinin baş harfleri büyük hale getirilmiştir.</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">Açıklama: Karşılaştırma işaretleri değiştirilerek Python'a uygun bir hale getirilmiştir.</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">Açıklama: null bloğu Python'a uygun olması için None bloğu olarak değiştirilmiştir.</td></tr>
</table>

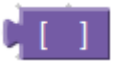
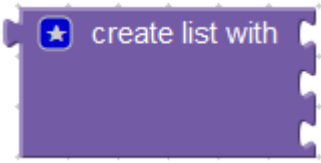
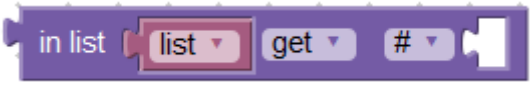
A.5. Metin Blokları

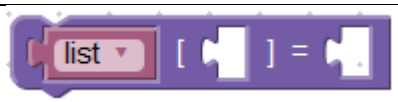
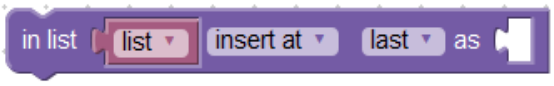
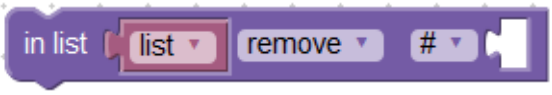
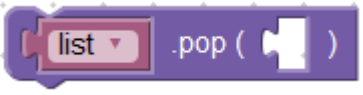
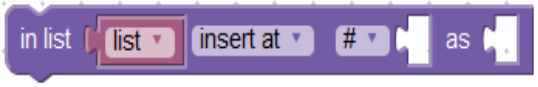
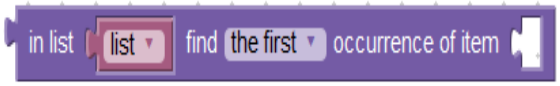
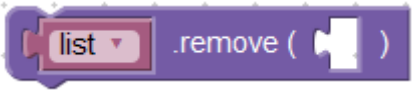
<table>
<tr><th>Eski</th><th>Yeni</th></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td colspan="2">Açıklama: Metin girmek için kullanılan bloktaki tırnak işareti Python'a uygun olarak değiştirilmiştir.</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td colspan="2">Açıklama: Kullanıcıdan bir değer girmesi için kullanılan blok Python'a uygun olarak değiştirilmiştir.</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td colspan="2">Açıklama: Metni büyük harf, küçük harf ya da sadece ilk harfini büyük harfe çevirmek için kullanılan bloklar Python diline uygun olarak değiştirilmiştir.</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td colspan="2">Açıklama: Metnin uzunluğunu bulmak için kullanılan blok Python'a uygun olarak değiştirilmiştir.</td></tr>
<tr><td colspan="2">  </td></tr>
<tr><td colspan="2">Açıklama: string, integer ve float veri tipleri arasında tip dönüşümü yapabilmek için yeni bir blok oluşturulmuş ve Blockly'e eklenmiştir.</td></tr>
</table>

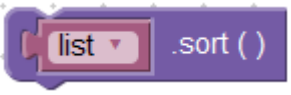


Açıklama: Yukarıdaki blokların çalışmada kullanılması gerekli olmadığı için ve yazılımın daha sade olması için çıkarılmasına karar verilmiştir.

A.6. Liste Blokları

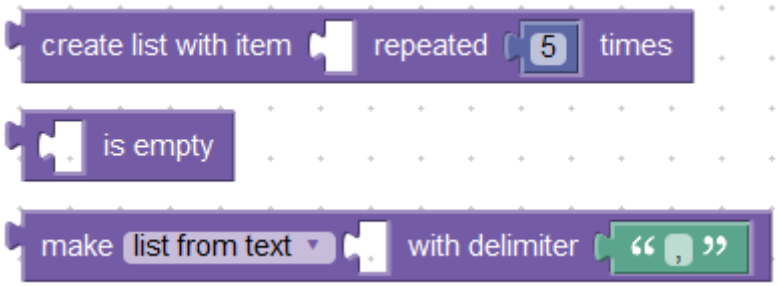
Eski	**Yeni**
	
Açıklama: Boş liste oluşturmak için kullanılan blok Python'a uygun olarak değiştirilmiştir.	
	
Açıklama: Eleman içeren bir liste oluşturmak için kullanılan blok Python sözdizimine uygun olarak değiştirilmiştir.	
	
Açıklama: Listeden bir eleman çağırmak için kullanılan blok Python sözdizimine uygun olarak değiştirilmiştir.	

	
Açıklama: Liste içeriğini değiştirmek için kullanılan blok Python sözdizimine uygun olarak değiştirilmiştir.	
	
Açıklama: Liste sonuna eleman eklemek için kullanılan blok Python sözdizimine uygun olarak değiştirilmiştir.	
	
Açıklama: Listenin belirlenen konumunda yer alan elemanı silmek için kullanılan blok Python sözdizimine uygun olarak değiştirilmiştir.	
	
Açıklama: Listenin belirlenen konumları arasındaki bir dilimi almak için kullanılan blok Python sözdizimine uygun olarak değiştirilmiştir.	
	
Açıklama: Listenin belirlenen konumuna eleman eklemek için kullanılan blok Python sözdizimine uygun olarak değiştirilmiştir.	
	
Açıklama: Listedeki bir elemanın konumunu bulmak için kullanılan blok Python sözdizimine uygun olarak değiştirilmiştir.	
	
Açıklama: Liste uzunluğunu bulmak için kullanılan blok Python'a uygun olarak değiştirilmiştir.	
	
Açıklama: Listedeki bir elemanı silmek için yeni bir blok oluşturulmuş ve Blockly'e eklenmiştir.	

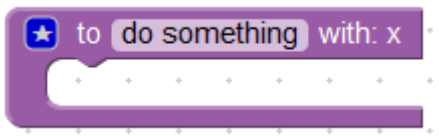
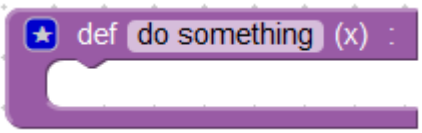
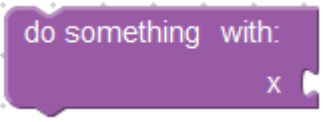
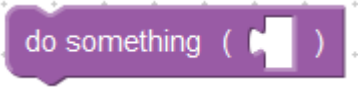
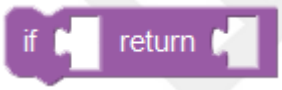
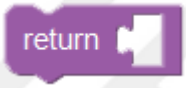

Açıklama: Listede bir elemanın olup olmadığını bulmaya yarayan yeni bir blok oluşturulmuş ve Blockly'e eklenmiştir.

Açıklama: Liste elemanlarını küçükten büyüğe ya da alfabetik bir biçimde sıralayan yeni bir blok oluşturulmuş ve Blockly'e eklenmiştir.

Açıklama: Listeyi tersine çevirmek için kullanılan yeni bir blok oluşturulmuş ve Blockly'e eklenmiştir.

Açıklama: Listede bir elemandan kaç tane olduğunu bulmaya yarayan yeni bir blok oluşturulmuş ve Blockly'e eklenmiştir.

Açıklama: Belirli bir aralıkta iterasyon yapmak için yeni bir blok oluşturulmuş ve Blockly'e eklenmiştir.

Açıklama: Yukarıdaki blokların çalışmada kullanılması gerekli olmadığı için ve yazılımın daha sade olması için çıkarılmasına karar verilmiştir.

A.7. İşlev Blokları

Eski	Yeni
	
Açıklama: İşlev tanımlama bloğu Python sözdizimine uygun bir şekilde değiştirilmiştir.	
	
Açıklama: İşlev çağırma bloğu Python sözdizimine uygun bir şekilde değiştirilmiştir.	
	
Açıklama: Orijinal Blockly versiyonunda *return* ifadesi sadece if karşılaştırmaları ile kullanılabiliyor, tek başına kullanılamıyordu. Araştırmacılar bu kısıtlılığı ortadan kaldırmış ve *return* ifadesinin tek başına da kullanılabilmesini sağlamıştır.	

B-) Sözdizimi ile İlgili Yapılan Değişiklikler

Blockly'nin orijinal versiyonunun Python dili sözdizimlerinde bir takım gereksiz ve sade olmayan kodlar bulunmaktadır. Araştırmacılar Blockly'nin Python sözdizimini çalışma için en uygun hale getirmişlerdir. Sözdizimi ile ilgili yapılan değişiklikler eski ve yeni halleri karşılaştırılarak verilmiş ve gerekli açıklamalar yapılmıştır.

Eski	Yeni
`a = None` `a = 5`	`a = 5`
Açıklama: Blockly'de değişken tanımlarken Python sözdizimi olarak yukarıda görüldüğü üzere *a=None* gibi gerekli olmayan kod blokları bulunmaktaydı. Bu kod bloğunun Python için gerekli olmadığı için çıkarılmasına karar verilerek daha sade bir görünüm kazandırılmıştır.	

```
a = None

def text_prompt(msg):
  try:
    return raw_input(msg)
  except NameError:
    return input(msg)


a = text_prompt('İsim giriniz: ')
```

```
a = input('İsim giriniz: ')
```

Açıklama: Blockly'de kullanıcıdan veri girmesi istendiğinde Python sözdizimi olarak yukarıda görüldüğü üzere gereksiz ve karmaşık kod blokları bulunmaktaydı. Bu yüzden bu kod bloğu çıkartılarak yerine daha sade bir Python kod bloğu konulmuştur.

```
liste = None

def first_index(myList, elem):
  try: theIndex = myList.index(elem) + 1
  except: theIndex = 0
  return theIndex


print(first_index(liste, 'a'))
```

```
print(liste.index('a'))
```

Açıklama: Blockly'de listedeki bir elemanın konumunu bulmak için yukarıda görüldüğü gibi gereksiz ve karmaşık bir Python kodu üretilmekteydi. Bu durum araştırmacılar tarafından düzeltilmiş ve oldukça sade bir biçime getirilmiştir.

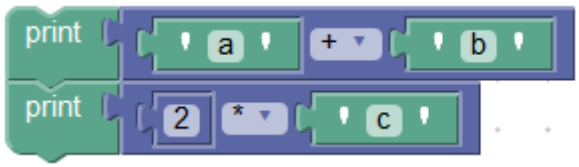
```
def upRange(start, stop, step):
  while start <= stop:
    yield start
    start += abs(step)

def downRange(start, stop, step):
  while start >= stop:
    yield start
    start -= abs(step)


a = 5
for i in (1 <= float(a)) and upRange(1, float(a), 1) or downRange(1, float(a), 1):
  print(i)
```

```
a = 5
for i in range(1,a):
  print(i)
```

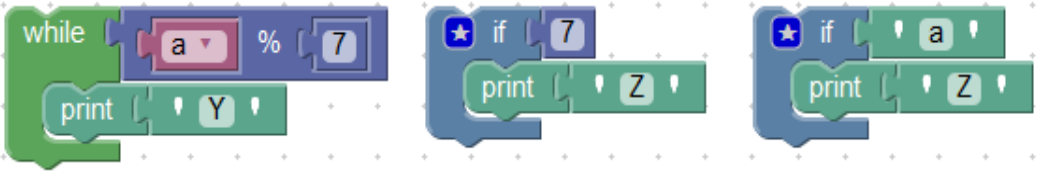
Açıklama: Blockly'de *for* döngüsü için sol üstteki kod bloğu üretiliyordu ve bu kod bloğu oldukça karışık ve gereksizdi. Araştırmacılar bunu sadeleştirerek sağ üstteki hale getirmişlerdir.

C-) Çalışma Prensibi ile İlgili Yapılan Değişiklikler

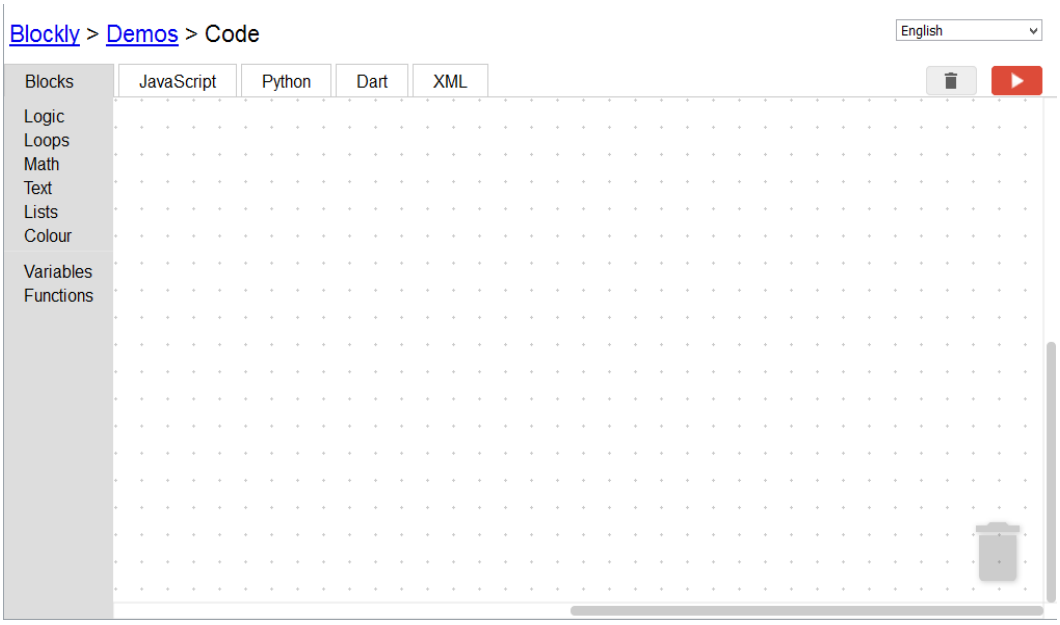
Araştırmacılar çalışma için daha uygun olması açısından Blockly'nin çalışma prensibinde birtakım değişikliklere gitmişlerdir. Yapılan bu değişiklikler açıklamalarıyla birlikte aşağıda verilmiştir.



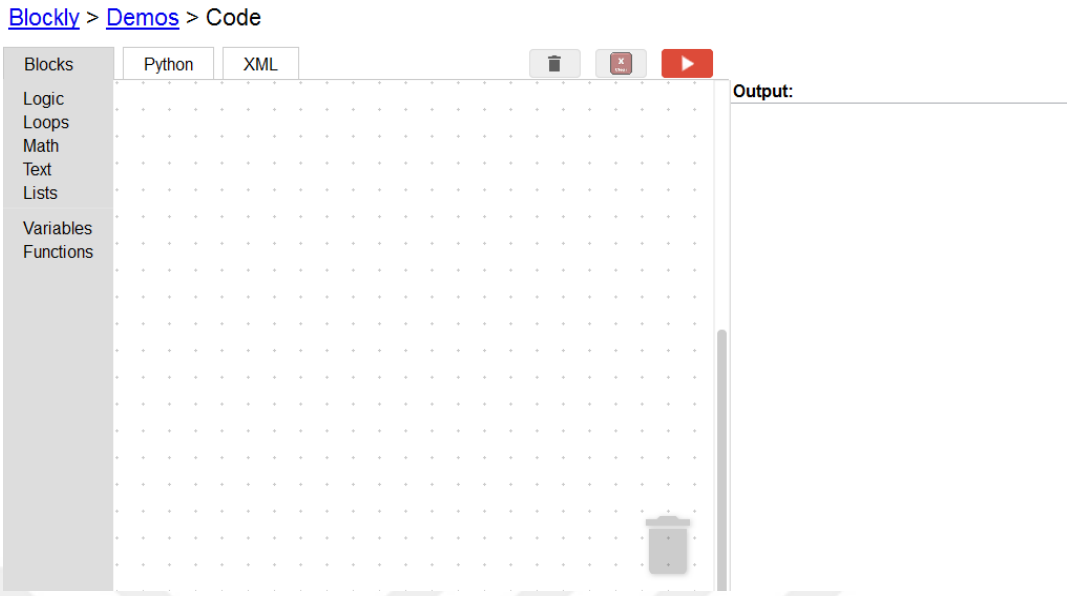
Açıklama: Blockly metinlerin toplanmasını ve bir sayı ile bir metnin çarpılmasını desteklemiyordu. Metin ve sayılar arasında işlem, programlama müfredatında yer alan bir konu olduğundan bu durumun düzeltilmesine karar verilmiş ve araştırmacılar tarafından şekilde görüldüğü üzere düzeltilmiştir.



Açıklama: Blockly *while* ve *if* yapılarında sadece *"if (a<b)"* gibi karşılaştırma ifadelerini destekliyordu. Örneğin *"if(7)"* ya da *"while(a%7)"* gibi ifadeleri desteklemiyordu. Araştırmacılar bunun önemli bir eksiklik olduğu düşüncesiyle bu durumu düzeltmişler Blockly'nin desteklemesini sağlamışlardır.



Blockly Eski Arayüzü



Blockly Yeni Arayüzü

Açıklama: Yukarıda Blockly'nin eski ara yüzü ve araştırmacılar tarafından düzenlenmiş yeni ara yüzü görülmektedir. Buradaki en önemli fark ekrana *Output* panelinin eklenmiş olmasıdır. Blockly'nin eski versiyonunda program çıktıları pop-up (aniden beliren) pencereler ile gösteriliyordu. Birden fazla çıktı olduğunda peşpeşe pop-up pencereleri çıkıyordu. Bu durum kullanıcıların çıktıları takip etmelerini zorlaştırıyordu. Çözüm olarak araştırmacalar tarafından program çıktılarını gösteren bir panel oluşturuldu. Bu özellik Blockly'nin gerçek programlama ortamlarına daha fazla benzemesine olanak tanımıştır.

EK 5: ETİK KURUL BELGESİ

T.C.
ONDOKUZ MAYIS ÜNİVERSİTESİ
KLİNİK ARAŞTIRMALAR ETİK KURULU

Sayı: B.30.2.ODM.0.20.08/1769 - 29.06.2015

Sayın Prof. Dr. Mustafa BAŞER

Etik Kurulumuza sunmuş olduğunuz **Görsel Programlama Öğretiminin Öğrencilerin Bilgisayar Programlamayı Öğrenmesine Ve Programlamaya Karşı Tutumlarına Etkisinin İncelenmesi** başlıklı OMÜ KAEK 2015/265 Karar nolu Anket çalışması nitelikli araştırma projeniz amaç, gerekçe, yaklaşım ve yöntemle ilgili açıklamaları, Klinik Araştırmalar Etik Kurulu yönergesine göre incelenmiş ve etik açıdan bir sakınca olmadığına, çalışmanın süresi 6 ayı geçerse 6 aylık bildirimlerinin yapılmasına, çalışma tamamlandıktan sonra sonucunun tarafımıza en geç üç(3) ay içerisinde bildirilmesine 28.05.2015 tarihli Etik kurulumuzda oy birliği ile karar verilmiştir.

Bilgilerinize arz/rica ederim.

Prof. Dr. A.Tevfik SÜNTER
Klinik Araştırmalar Etik Kurulu Başkanı

Ondokuz mayıs Üniversitesi Tıp Fak. Klinik Araştırmalar Etik Kurulu Tel:(0362)3121919/2782 -4576007 Omutaek@gmail.com
Hastane içi 1.Kat (Özel servis karsısı) Atakum/SAMSUN

EK 6: PROGRAMLAMA YETENEK TESTİ KULLANIM İZNİ

Kimden: "Bruce Woodcock" <B.E.Woodcock@kent.ac.uk>
Kime: "Mustafa Baser" <mustafa.baser@omu.edu.tr>
Gönderilenler: 14 Aralık Çarşamba 2011 16:03:28
Konu: Re: Computer Programming Aptitude Test

Hello Mustafa, this is fine as long as the test is not uused commercilally

All we require is that acknowledgement of our ownership is clearly given and links back to the relevant pages on our site are made if appropriate.

Something like:

Reproduced with permission of the University of Kent Careers Advisory Service
www.kent.ac.uk/careers who own copyright.

Contact Bruce Woodcock bw@kent.ac.uk for details.

I hope your research goes well.

Best wishes,
--

Bruce

Bruce Woodcock
Careers Advisory Service,
Keynes College Driveway
University of Kent,
Canterbury,
Kent CT2 7ND, England
Tel: 01227 764000 ext. 7594
Email: B.E.Woodcock@kent.ac.uk
www.kent.ac.uk/careers

On 13/12/2011 11:59, Mustafa Baser wrote:
Dear Bruce Woodcock,

Thank you for quick reply. I want to translate the test into Turkish and use it in my research. Would you give permission to me to use the test.

Sincerely yours,
Mustafa

----- Orijinal Mesaj -----
Kimden: "Bruce Woodcock" <B.E.Woodcock@kent.ac.uk>
Kime: "Mustafa Baser" <mustafa.baser@omu.edu.tr>
Gönderilenler: 12 Aralık Pazartesi 2011 15:57:48
Konu: Re: Computer Programming Aptitude Test

Hello Mustafa,

There is no research which has been carried out on this. The test is based on a number of skills known to be required by successful programmers.

Best wishes,

ÖZGEÇMİŞ

Mehmet Fatih Yiğit 29.11.1989 tarihinde Bursa'da doğdu. Sinop Anadolu Teknik Lisesi'ni bitirdikten sonra Boğaziçi Üniversitesi Eğitim Fakültesi'nden 2012 yılında mezun oldu. Mezuniyetinden bu yana araştırma görevlisi olarak görev yapan Yiğit, iyi derecede İngilizce bilmektedir. Temel ilgi alanları, programlama eğitimi ve görsel programlama dilleridir.

İletişim Bilgileri

E mail : mehmetfatihyigit57@gmail.com

Telefon : 0546 926 8920

Bildiriler / Yayınlar

Yiğit, M. F. ,Başer, M. (2015). Learning Difficulties and Use of Visual Technologies in Learning to Program, *Participatory Educational Research, 2015-II*, 27-34.