T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN-EDEBİYAT FAKÜLTESİ
MATEMATİK BÖLÜMÜ

PYTHON İLE PROGRAMLAMA

Hazırlayan

ASLIHAN NUR PEHLİVAN

16025043

Matematik Bölümünde Hazırlanan

LİSANS BİTİRME TEZİ

Tez Danışmanı: Dr.Öğr.Üyesi Elif TARIM

İSTANBUL, 2021

ÖNSÖZ

Bütün tez çalışmam sırasında bana bilgi ve birikimiyle yardımcı olan; tez konumun seçiminde ve çalışmamın gidişatında her zaman ilgili olan değerli tez danışman hocam Sayın Dr.Öğr.Üyesi Elif Tarım'a sonsuz teşekkürlerimi ve saygılarımı sunarım.

Tüm eğitim hayatım boyunca maddi ve manevi her konuda yanımda olan başta annem olmak üzere tüm aileme çok teşekkür ederim.

Ocak, 2021

Aslıhan Nur Pehlivan

İÇİNDEKİLER

Sayfa

ŞEKİL LİSTESİ

ÇİZELGE LİSTESİ

ÖZET

Bu tezde Python Programlama Dili'nin anlatımı ele alınmıştır. Bu programlama dili, açık kaynaklı, platform bağımsız, hızlı ve dinamik bir dildir. Ayrıca çok geniş kütüphanelere ve yalın bir yapıya sahip olduğundan kullanımı diğer dillere göre daha kolaydır. Bu özelliklerinden dolayı Python programlama dili; veri bilimi, veri analizi, robot programlama ve yapay zeka gibi güncel teknolojiler için oldukça tercih edilmektedir. Bunların yanı sıra web uygulamalarında da sıkça kullanılmaktadır.

Anahtar Kelimeler: Python Programlama Dili, Değişkenler, Operatörler, Seçimli Yapılar, Döngüler, Fonksiyonlar, Karakter Dizileri, Liste, Demet, Küme, Sözlük

1. PYTHON HAKKINDA

1.1 Python Nedir?

Python programlama dili C++, C#, JavaScript veya Java gibi programlama dillerine benzeyen kullanışlı bir programlama dilidir. Bu programlama dili ilk olarak 90'lı yılların başında Guido Van Rossum tarafından Hollanda'da tasarlandı. Sanılanın aksine bu programlama dili adını piton yılanından almamaktadır. Guido'nun favori programlarından biri olan "Monty Python's Flying Circus (Monty Python'ın Uçan Sirki)" adlı komedi dizisinden almaktadır. Guido'nun böyle bir dili oluşturmadaki başlıca sebeplerinden biri, 80'li yıllarda çoğu programlama dilinin çok ciddi ve karmaşık olmasıydı. Guido, o kadar da zor olmayan, eğlenceli ve kullanımı keyifli aynı zamanda da güçlü bir programlama dili üretmeyi hedeflemiştir.

Özellikle son yıllarda Python programlama diline olan ilgi bir hayli artmıştır ve bu ilginin oluşmasındaki başlıca sebeplerden biri öğreniminin kolay, kullanımının hızlı olmasıdır.

Python dili öğrenme kolaylığının yanında internet ortamında ücretsiz paylaşılan ve güncellenen geniş kütüphanesi/modülleri sayesinde günümüzde robot programlamadan veri analizine kadar çok farklı alanlarda tercih edilmektedir [1].

Diğer programlama dillerinden farklı olarak Python dilinde yazılan kodlarda girintilerin önemi oldukça büyüktür. Python, kodun nasıl yerleştirildiğine çok duyarlıdır ve en ufak bir boşluk ifadesi bile programda hata oluşmasına sebep olabilir.

1.2 Python Sürümleri

Python programlama dilinin piyasaya çıkışından bu yana pek çok Python sürümü kullanıldı. Farklı Python serilerinin var olmasından ötürü, Python ile program yazarken hangi seriye ait sürümlerden birini kullandığınızı bilmeniz, yazacağınız programın kaderi açısından büyük önem taşır [4].

Günümüzde ise Python programlama dilinin, Python2 ve Python3 olmak üzere iki adet versiyonu bulunmaktadır [2]. İki versiyon arasındaki fark basitçe şu şekilde ifade edilebilir:

- Python 2'de şu şekilde yazılır; print 'Hello World'
- Python 3'de şu şekilde yazılır; print ('Hello World')

Bu iki yazım arasındaki fark çok büyük görünmüyor olsa da '()' işaretinin kullanımı büyük bir

fark oluşturur ve Python versiyonlarından birinin çalıştırdığı kod muhtemelen diğer versiyonda çalışmayacaktır.

Peki bu durumda hangi versiyon öğrenilmeli?

Python programlama diline yeni başlanıyorsa Python 3'ün öğrenilmesi daha iyi olacaktır çünkü çeşitli sektörlerde Python 3' e olan ilgi ve merak hızla artmaktadır. Aynı zamanda Python 2 zamanla tedavülden kalkacaktır. Fakat günümüzde pek çok organizasyon halen Python 2 kullanmaktadır. Çalışılacak projede Python 2 kullanışmışsa ve yardımcı modüller Python 3 'e aktarılmamışsa Python 2 kullanılması gerekebilir. Bu tezde Python 3 anlatılacak ve Python yazıldığında Python 3 kastedilecektir.

1.3 Python Programını Çalıştırma

Python programını çalıştırmanın birkaç yöntemi vardır. Bunlardan bazıları;

- Python yorumlayıcısını etkileşimli olarak kullanarak çalıştırma
- Bir dosyada depolama ve Python komutunu kullanarak çalıştırma
- Komut dosyası içinde kullanılacak Python yorumlayıcısını belirten bir komut dosyası olarak çalıştırma
- Bir Python derleyicisi IDE (Integrated Development Enviroment-Entegre Yazılım Geliştirme Ortamı) içinden çalıştırma [2]

şeklinde sıralanabilir.

1.3.1 Derleyici ile Yorumlayıcının Farkı

Yorumlayıcıda kodun okunması ve çevrilmesi programın çalışması sırasında yapıldığından hız düşüktür. Ayrıca yorumlayıcı yalnızca karşılaştığı ilk hatayı rapor edebilir. Bu hata düzeltildiğinde sonraki çalışmada da program ancak bir sonraki hataya kadar ilerleyebilir. Oysa derleyici kaynak kodundaki bütün hataları bulabilir. Buna karşılık hata ayıklama işlemi yorumlayıcılarda daha kolaydır. Ayrıca derleyicilerle gelen bazı sınırlamaların kalkması nedeniyle daha esnek bir çalışma ortamı sağlanır. Son yıllarda iki yöntemin üstün yanlarını birleştiren karma diller (Java ve Python gibi) öne çıkmaya başlamışlardır.

Derleyici programın tamamını kontrol eder, bir satırda hata varsa program satırları bittikten sonra hatayı gösterir. Yani kaynak kodu bir defa çevirir. Yorumlayıcı ise programın bir yerinde

hata varsa çalışırken hatalı satıra geldiğinde durur ve hata mesajı verir. Hata düzeldikten sonra tekrar birinci satırdan itibaren kodu çevirir [1].

1.4 Python Editörleri (IDE)

Bir programlama dili ile kod yazmak, derlemek, çalıştırmak, hataları ayıklamak için kısaca IDE adı verilen editörler kullanılır. Resmi Python editörü IDLE haricinde PyDev, PyCharm, Geany, Eric, Thonny gibi birçok farklı editör geliştirilmiştir [1].

IDLE (Integrated Development and Learning Enviroment), Python kurulumu ile gelen resmi kod editörünün ismidir. Özellikleri;

- %100 Python dilinde kodlanmıştır.
- Platform bağımsızdır, yani Windows, Linux, Unix, Mac OS X ve RaspberryPi gibi tüm işletim sistemlerinde kullanılır.
- Giriş, çıkış komutları, hata mesajları farklı renklerde gösterilmektedir.
- Akıllı kod tamamlama özelliğine sahiptir.
- Ek kurulum gerekmez, Python kurulduğunda kurulur.

Pydev, Eclipse için geliştirilmiş bir Python eklentisidir. Özellikle Java ve C++ programcıları Eclipse editörünü tercih ederler.

Geany, hızlı ve hafif (az yer kaplayan) bir editördür. Geany GTK kütüphanesi tarafından desteklenen her (Linux, Mac OS X ve Windows) ortama kurulabilir.

PyCharm, JetBrains firması tarafından geliştirilen başka bir Python editörüdür. Bu editörde programın çıktısı ekranın altındaki bir çıktı konsolunda gösterilir.

Visual Studio Code, Microsoft firması tarafından geliştirilen bu editör ile Python programları da derlenip, çalıştırılabilir. Windows, MacOS ve Linux ortamlarında kullanılabilen hafif ama güçlü bir editördür [1].

Spyder, sevilen ve sıkça kullanılan bir başka Python editörüdür. Gelişmiş düzeyde düzenleme, hata ayıklama ve veri keşfetme özelliği sunar.

2. PYTHON PROGRAMLAMA

2.1 Python Yorum Satırları

Yorumlar, program çalıştırılırken derleyicinin görmezden geldiği bölümlerdir. Yazılan kodları başkalarının da rahatlıkla anlayabilmesi için, programın yorumlarla desteklenmesi tavsiye edilir. Elbette programınızı yorumlarla desteklemeseniz de programınız sorunsuz bir şekilde çalışacaktır. Ama programı yorumlarla desteklemek büyük kolaylık sağlar. Python'da yorumlar # işareti ile gösterilir.

```
#Bu bir yorum satırı
>>>name = input('Enter your name: ')
>>>print(name) #Bu da satırın sonunda bir yorum
```

2.2 Merhaba Dünya

Bir programlama dilini öğrenmeye başlarken genellikle ilk olarak *"*Merhaba Dunya*"* yazılır. Basitçe "Merhaba Dunya" karakter katarının (string) ekrana yazdırılması aşağıdaki gibi ele alınır.

```
>>> print('Merhaba Dunya')
Merhaba Dunya
```

Şimdi bu programı ve print fonksiyonunu yakından inceleyelim. Herhangi bir IDE'de Python dosyası oluşturduğumuzda *.py* uzantılı bir dosya oluşur. Örneğin, yukarıdaki *print()* fonksiyonunu içeren *hello.py* isimli bir dosya oluşturabiliriz [2].

Peki print() fonksiyonu nereden gelir?

print() fonksiyonu bir şeyleri yazdırmayı sağlayan, önceden tanımlanmış bir fonksiyondur (built-in function). Önceden tanımlanmış derken kastedilen fonksiyonun Python ortamında yerleşik olduğu ve Python yorumlayıcısı tarafından anlaşılmış olduğu anlamına gelir. Çıktı, çıktı akışı olarak bilinen bölgeye yazdırılır. Bu, harfler ve sayılar gibi bir veri akışını (dizisini) işler [2].

printf() fonksiyonu girdi olarak ne verildiyse, çıktı olarak onu verir. Örneğin; string girildiğinde çıktı olarak yine string verir (Tıpkı "Merhaba Dunya" uygulamasında olduğu gibi). 42 gibi bir integer değişken girilmişse, çıktı olarak yine olarak 42'yi yazdırır.

Şimdi uygulamamızı biraz daha ilginç bir hale getirelim. Uygulamamız isim sorsun ve bireysel olarak "Merhaba Dunya" desin. Bunun için;

```
>>>print('Merhaba Dunya')
>>>user_name = input('Adinizi giriniz: ')
>>>print('Merhaba ', user_name)
```

Bu durumda programın çıktısı şu şekilde olur;

```
Merhaba Dunya
Adinizi giriniz: Aslihan
Merhaba Aslihan
```

Burada *input()* fonksiyonu, kullanılmak üzere kullanıcıdan string tipinde bir girdi alır. Fonksiyonun içerisinde yer alan string bir bilgi istemi olarak gösterilir ve kullanıcıdan girdi alana kadar dönüş anahtarını bekleyecektir. Bu fonksiyon tıpkı *print()* fonksiyonunda olduğu gibi Python dilinin içerisinde yer alan önceden tanımlanmış bir fonksiyondur.

Yukarıda verilen örnekte *input()* fonksiyonun almış olduğu string değeri *user_name* adlı değişkende depolanmış olur.

Değişken; string, sayılar, boolean gibi verileri tutmak için kullanılabilen bilgisayar belleğinin adlandırılmış bir alanıdır. Bu durumda, *user_name* değişkeni, kullanıcı tarafından girilen stringi tutacak bir bellek alanı için bir etiket görevi görür. Böylece *user_name* değişkeni bu bellek alanına kolay ve rahat bir şekilde erişmemizi sağlar [2]. Kullanıcı tarafından girilen ifadeyi başka bir ifadede kullanmak istiyorsak bunu yalnızca değişkenin etiketine atıfta bulunarak yapabiliriz. Yukarıda verilen örnekte bellekte tutulan değeri yeniden kullanmak istediğimiz için girilen değeri 3.satırda *user_name* etiketi ile yeniden çağırdık.

"print()" fonksiyonu kullanılırken oluşacak çıkış mesajının biçimi çeşitli şekillerde ayarlanabilir. Bunlara bazı örnekler verilebilir;

- **print('p', 'y', 't', 'h', 'o', 'n')** : Her bir karakter arasına boşluk bırakır.
- Ekran çıktısı*: p y t h o n*

- **print('p', 'y', 't', 'h', 'o', 'n', sep=',')** : sep (ayraç) fonksiyonu ekrana basılacak her değerin arasına belirtilen karakteri koyar. Ekran çıktısı: *p,y,t,h,o,n*
- **print("python",end='?')** : end fonksiyonu ekrana basılan değerin sonuna belirtilen karakteri koyar. Ekran çıktısı: *python?*

2.3 Değişkenler ve Değer Ataması

Bilgilerin geçici olarak tutulduğu yere bellek denir. Değişkenler de bilgisayar belleğinde tutulur/saklanır. [1]

İçerisinde veri sakladığımız, ismini ve tipini bizim belirlediğimiz bellek alanlarına değişken (variable) adı verilmektedir [1]. Değişkenler, bellekte herhangi bir adresi gösteren sembolik birer isimdir. Değerlendirilmiş bir ifadenin sonucu daha sonra programda kullanılmak istenirse bu, bir değişkene kaydedilerek yapılabilir. Kullanılan programlama dili ne olursa olsun bir programda kullanılan değişkenlerin isimleri programcı tarafından belirlenmektedir, ama değişkenin belleğin neresinde saklanacağına (adres bilgisine) ise bilgisayar kendisi karar verir.

Bir değişken kullanılmadan önce program içerisinde istenilen her yerde tanımlanabilir. Değişkenler tanımlanırken bir atama ifadesinin kullanılmasıyla değerleri depolarlar. Aritmetiksel veya sözel bir ifadeyi bir değişkene aktarmak/atamak için '=' operatörü kullanılır. Bir atama ifadesi bir değişken adı, atama operatörü ve depolanacak değerden oluşur.

```
>>>toplam=0
>>>ad="Aslihan"
```

Daha önce vermiş olduğumuz Merhaba Dünya uygulaması örneğinde *user_name=input('Enter your name: ')* ifadesi de bir atama ifadesidir. Atama operatörü Python'da kullanılan en yaygın operatörlerden biridir.

Değişkenlere değer aktarırken değişkenin eşitliğin sol tarafında olduğuna dikkat edilmelidir. Ayrıca programlamaya yeni başlayanlar "=" ile "==" operatörlerini sıklıkla birbirine karıştırmaktadırlar. "=" operatörü belirttiğimiz gibi eşitliğin sağ tarafındaki ifadenin sonucunun, sol taraftaki değişkene aktarılmasını sağlar. "==" operatörü ise "eşit mi?" sorgusunu yapmayı sağlar. Yani "=" bir atama operatörü iken "==" bir karşılaştırma

operatörüdür.

2.3.1 Değişkenlere İsim Verme Kuralları

Python programlama dilinde, değişken adı olarak belirleyebileceğimiz kelime sayısı neredeyse sınırsızdır. Yani hemen hemen her kelimeyi değişken adı olarak kullanabiliriz. Ama yine de değişken adı belirlerken dikkat etmemiz gereken bazı kurallar var. Bu kuralların bazıları zorunluluk, bazıları ise yalnızca tavsiye niteliğindedir. [4]

Değişken tanımlarken dikkat edilmesi gereken hususlar;

- Bir değişkenin ilk karakteri harf veya alt tire olmalıdır.
- Değişken isimlerinin içerisinde alt çizgi alt tire hariç, boşluk veya diğer özel karakterler bulunmamalıdır. (.,?*:;!/()-+%=”^#... karakterleri gibi)
- Python büyük-küçük harf ayrımı yapan (Case sensitive) bir dildir. (Yani *ad* değişkeni ile *Ad* değişkeni farklı değişkenler olarak kabul edilir.)
- Değişken isimleri, kullanılan programlama diline ait komutları içeremez. O programlama diline ait komutlara ayrılmış sözcükler (reserved words) veya anahtar sözcükler (keywords) de denilmektedir. Çizelge 2.1.’de yer alan ayrılmış sözcükler değişken ismi olarak kullanılamaz.

Çizelge 2.1. Python dili komutları/anahtar kelimeleri (keywords)

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Değişkenlere (aynı zamanda nesne ve sınıf isimleri gibi diğer tanımlayıcılara da) isim verirken, verilen ismin içerikle anlamlı olmasına dikkat edilmelidir. İyi bir değişken ismi içerisinde bulanan veri ile ilişkili olmalıdır. Bu isimlendirmeye mnemonic (anımsatıcı) teknik adı verilir. Python mnemonic bir isim seçip seçmediğinize dikkat etmez. Programlama dili için *xkfghjprf*

değişken ismi ile, *xkfgajprf* değişken ismi birbirinden kesinlikle farklıdır. Fakat kullanıcının bu iki kelime arasındaki ufak farkı fark etmesi oldukça zordur. Bu nedenle içerik ile ilişkili değişken isimlendirmesi yapmak, kodun anlaşılabilirliği açısından oldukça önemlidir.

2.4 Veri Tipleri

Python'da diğer dillerin aksine değişken tanımlarken veri tipinin belirtilmesine gerek yoktur. Fakat her değişkenin belirli bir veri tipine sahip olduğu (bool, int, float, complex, str gibi) unutulmamalıdır. Python veri tipi tanımlamasını kendisi otomatik yaparak kod yazımını basitleştirmektedir [1].

- **Str Veri Tipi:** Python'da string, iki adet tek tırnağın veya çift tırnağın içerisinde bulunan sıralanmış karakter dizileridir. Klavyede tek bir tuşa basarak yazabilen her şey bir karakter belirtir. Örneğin; bir 'a' harfi, bir '2' rakamı, '\', '[', '$' gibi özel karakterler veya bir boşluk karakter belirtir.

```
>>>some_string =’ ’
```

- **Sayısal Veri Tipleri:** Sayısal veriler ise üç farklı veri tipinde tutulur. Tamsayılar için *int*, kesirli sayılar için *float* ve karmaşık (kompleks) sayılar için *complex* olmak üzere üç adet sayısal veri tipi kullanılır.

- **Bool Veri Tipi:** Mantıksal (iki durum karşılaştırması gibi) sorgulamalarda *bool* veri tipi kullanılır. Boolean tipinde bir değişken sadece *True* ve *False* değerlerini alabilir. Burada kelimelerin büyük harf ile başladıklarına dikkat edilmelidir, true ve false şeklinde yazılırsa Python bu kelimeleri olması gerektiği şekilde algılayamaz.

2.4.1 type() fonksiyonu

Python'da bir değişkenin veya fonksiyonun veri tipini öğrenmek için type() fonksiyonu kullanılır.

```
>>> type("Elma")
<class 'str'>
```

```
>>> type(34657)
<class 'int'>

>>> type("34657")
<class 'str'>
```

2.4.2 Veri Tipi Dönüşümü

Değişkenlere değer aktarımında, veri tipine bağlı olarak alabileceği değerlere ve tip uyumuna dikkat etmek gerekir. Python, tip dönüşümü için yerleşik (built-in) fonksiyonları kullanır. Bunlar [1];

- int(): string veya float veri tipini int (tamsayı) veri tipine dönüştürür.
- float(): string veya int veri tipini float (kesirli sayı) veri tipine dönüştürür.
- str(): ifadeyi string veri tipine dönüştürür.

```
>>> pi=int(3.14)
>>> print(pi)
3

>>> PI=3.14
>>> print(“Pi:” + str(PI))
Pi:3.14
```

Bir Python programında “input()” fonksiyonu ile kullanıcından bir sayısal değer alındığında programın bu değeri string olarak alacağına dikkat edilmelidir. Eğer girilen değer programda sayısal olarak kullanılacaksa, kullanılmadan önce mutlaka integer veya float veri tipine dönüştürülmelidir.

```
asli=input("Aslı'nın yaşını giriniz:")
fatma=input("Fatma'nın yaşını giriniz:")
print("Aslı ve Fatma’nın yaşları toplamı:",asli+fatma)
```

Verilen kodda asli değişkenine 5, Fatma değişkenine ise 8 değeri girilmiş olsun. Bu durumda programın çıktısı:

```
Aslı'nın yaşını giriniz:5
Fatma'nın yaşını giriniz:8
Aslı ve Fatma’nın yaşları toplamı: 58
```

şeklinde olacaktır. Girilen değerler sayısal değerler olmadığı için program istenilen işlemi doğru bir şekilde gerçekleştirememiştir. Bu sorunu düzeltmek için girilen değerlerin integer tipine dönüştürülmeleri gerekmektedir. Program yeniden düzenlenirse;

```
asli=int(input("Aslı'nın yaşını giriniz:"))
fatma=int(input("Fatma'nın yaşını giriniz:"))
print("Aslı ve Fatma’nın yaşları toplamı:",asli+fatma)
```

Böylece kod istenilen işlemi doğru bir şekilde yapacaktır.

```
Aslı'nın yaşını giriniz:5
Fatma'nın yaşını giriniz:8
Aslı ve Fatma’nın yaşları toplamı: 13
```

3. OPERATÖRLER VE MATEMATİKSEL İFADELER

İngilizce'de *operator* adı verilen işleçler, sağında ve solunda bulunan değerler arasında bir ilişki kurar. Bir işlecin sağında ve solunda bulunan değerlere işlenen (*operand)* adı verilir [4]. Operatörlerle işlem yaparken işlem önceliğine/üstünlüğe ve çalışma sırasına dikkat edilmelidir.

Operatörleri 5 başlık altında inceleyeceğiz.

- Aritmetiksel Operatörler
- Atama Operatörleri
- Karşılaştırma Operatörleri
- Mantıksal Operatörler
- Aitlik Operatörleri

Python'da her bir operatörün bir fonksiyon eşdeğeri vardır. Örneğin 'a+b' yerine 'add.(a,b)' fonksiyonu kullanılabilir. Tabii fonksiyonlar operatör modülü içerisinde yer aldığından kullanmadan önce 'import operator' ile programa eklenmelidir [1].

3.1 Aritmetiksel Operatörler

Aritmetiksel operatörler, matematikte kullanılan ve sayılarla aritmetik işlemler yapmamızı sağlayan yardımcı araçlardır.

```
>>> 45 + 33
78
```

Burada *45* ve *33* değerlerine işlenen (*operand*) adı verilir. Bu iki değer arasında yer alan *+* işareti ise bir operatördür (*işleç*). Dikkat ederseniz *+* operatörü *45* ve *33* adlı işlenenler arasında bir toplama ilişkisi kuruyor.

Toplama (+) ve çarpma (*) operatörleri ile ilgili olarak dikkat edilmesi gereken bir durum daha vardır. Bu operatörler Python'da birden fazla anlama gelirler. Strginler ile birlikte kullanılmaları durumunda kullanım amaçları farklı olur.

```
>>> "Aslihan" + "nur"
'Aslihannur'
```

Yukarıdaki kullanımda görüldüğü üzere “+” operatörü stringleri birleştirme görevi görür. Benzer şekilde “*” operatörünün de çarpma işlemi dışında farklı bir kullanımı vardır.

```
>>> "hızlı " * 2
'hızlı hızlı '
```

Çizelge 3.1.’de tüm aritmetiksel işlem operatörleri, örnek kullanımları ve Python’daki fonksiyon eşdeğerleri görülmektedir.

Çizelge 3.1. Aritmetiksel işlem operatörleri

Operatör Tipi	**Operatör**	**Örnek Kullanım**	**Operatör Fonksiyon Eşdeğeri**
Toplama	+	a+b	operator.add(a,b)
Çıkarma	-	a-b	operator.sub(a,b)
Çarpma	*	a*b	operator.mul(a,b)
Bölme	/	a/b	operator.truediv(a,b)
Tamsayı Bölme	//	a//b	operator.floordiv(a,b)
Üs Alma	**	a**b	operator.pow(a,b)
Mod (Kalan bulma)	%	a%b	operator.mod(a,b)
Matris Çarpma	@	a@b	operator.matmul(a,b)

3.2 Atama Operatörleri

Aritmetiksel ya da sözel bir ifadeyi bir değişkene aktarmak/atamak için ‘=’ atama operatörü kullanılır. Bir atama operatörünün kullanım amacı basit olup, eşitliğin sağındaki ifadeyi soluna aktarmaktadır. Örneğin ‘a=3’ ifadesinde ‘a’ değişkenine ‘3’ değeri atanmış/aktarılmış olur [1].

Python dilinde matematiksel işlemlerde, bu ‘=’ atama operatörü ile aritmetiksel operatörlerin birleşmesi ile yeni operatörler oluşturulabilir ki; bu yeni operatörlere *aritmetiksel atama operatörleri* denir. Bu operatörlerin gösterimi, aritmetiksel operatör ve ‘=’ parametresinin yan yana birleştirilmesi (+=, *=) şeklindedir. Örneğin ‘+=’ operatörü şu şekilde kullanılabilir [1]:

```
>>> a += 5
>>> print(a)
28
```

Bu koda eşdeğer kod ise şu şekildedir;

```
>>> a = a + 5
>>> print(a)
28
```

Çizelge 3.2.'de aritmetiksel atama operatörleri, Python'daki gösterimleri ve işlem operatörleri olarak eşdeğerleri görülmektedir.

Çizelge 3.2. Aritmetiksel atama operatörleri

İşlem	Operatör	Python Gösterimi	Eşdeğeri/Açılımı
Aktar/Ata	=	A=B=3 #python dili çoklu atamaya izin verir	A=3 B=3
Topla ve aktar	+=	A+=B	A=A+B
Çıkar ve aktar	-=	A-=B	A=A-B
Çarp ve aktar	*=	A*=B	A=A*B
Böl ve aktar	/=	A/=B	A=A/B
Kalanı bul ve aktar	%=	A%=B	A=A%B
Tamsayı böl ve aktar	//=	A//=B	A=A//B

3.3 Karşılaştırma Operatörleri

Adından da anlaşıldığı üzere karşılaştırma operatörleri, işlenenler (*operands*) arasında bir karşılaştırma ilişkisi kuran operatörlerdir [4]. Verilerin karşılaştırılmasında kullanılırlar. Sonuç doğru ise *True*, yanlış ise *False* değerini üretirler. Bir karşılaştırmanın, koşulun söz konusu olduğu döngü veya karar yapılarında da bu karşılaştırma operatörlerinden biri mutlaka kullanılmalıdır.

```
>>>print(3<6)
True
>>>print(10>=24)
False
```

Çizelge 3.3.' de karşılaştırma operatörleri, kullanımları ve fonksiyon eşdeğerleri gösterilmiştir.

Çizelge 3.3. Karşılaştırma operatörleri

Operatör Tipi	**Operatör**	**Örnek Kullanım**	**Operatör Fonksiyon Eşdeğeri**
Büyük	>	a>b	operator.gt(a,b)
Küçük	<	a<b	operator.lt(a,b)
Büyük Eşit	>=	a>=b	operator.ge(a,b)
Küçük Eşit	<=	a<=b	operator.le(a,b)
Eşit	==	a==b	operator.eq(a,b)
Eşit Değil	!=	a!=b	operator.ne(a,b)

3.4 Mantıksal Operatörler

Mantıksal operatörler, birden fazla verinin birbiri ile kıyaslanması durumunda kullanılırlar. Kıyaslamanın sonucuna bağlı olarak da hangi işlemlerin yapılacağına karar verilir. Mantıksal operatörlerin sonucu, boolean tipi değişkenlerde tutulur ve sonuç olarak sadece birbirinin tersi olan iki değerden *'True/Doğru'* veya *'False/Yanlış'* biri üretilir [1]. Çizelge 3.4.'de mantıksal operatörlerin örnek kullanımı ve kullanım sonucunda oluşacak ekran çıktısı gösterilmiştir.

Çizelge 3.4. Mantıksal operatörler

İşlem	Operatör	Örnek Kullanım	Ekran Çıktısı
Mantıksal VE	and	>>>A=B=3 ; C=5 >>> (A<B) and (B<C)	False
Mantıksal VEYA	or	>>>(A<B) or (B<C)	True
Mantıksal DEĞİL	not	not(A<B)	True

Mantıksal operatörlerin işlevleri şu şekilde açıklanabilir;

- **And** Operatörü: Mantıksal (Lojik) VE (AND) işlemi operatörüdür, lojik ifadelerin her ikinin de doğru olması durumunda True sonucunu aksi takdirde False sonucunu üretir.
- **Or** Operatörü: Mantıksal VEYA (OR) işlemi operatörüdür, lojik ifadelerden herhangi birinin doğru olması durumunda True sonucunu, aksi takdirde False sonucunu üretir.
- **Not** Operatörü: Mantıksal DEĞİL(NOT) işlemi operatörüdür, lojik ifadenin değilini(tersini) alır. Örneğin; not(True)=False sonucunu üretir.

3.5 Aitlik Operatörleri

Aitlik operatörleri, bir karakter dizisi ya da sayının, herhangi bir veri tipi içinde bulunup bulunmadığını sorgulamamızı sağlayan operatörlerdir [4]. Ait olma durumuna göre *True* veya *False* sonucunu üretir. Aşağıda verilen örnekte ve Çizelge 3.5.'de aitlik operatörlerinin kullanımları gösterilmiştir.

```
>>> a = "abcd"
>>> "a" in a
True
```

Çizelge 3.5. Aitlik operatörleri

Operatör Tipi	Operatör	Örnek Kullanım
Ait olma	in	A in liste
Ait olmama	not in	A not in liste

3.6 Python Operatörlerinin Öncelik Sıraları

Programlama dillerinde bir ifadede birden fazla operatör göründüğünde, değerlendirme sıralaması öncelik kurallarına bağlıdır. Matematiksel operatörler için Python, matematiksel kuralı izler [3].

Operatörlerin genel öncelik sırası Çizelge 3.6.'da verilmiştir. Aynı önceliğe sahip işlemlerde soldaki işlem daha önce işlenir.

Çizelge 3.6. Python operatörlerin öncelik sıraları

Öncelik Sırası	**Operatör veya İşlem**	**Açıklaması**
En Yüksek	( )	Parantez içi
	pow(), sqrt(), min(), …	Fonksiyonlar
	**	Üs alma
	+x, -x	Pozitif, Negatif
	*, @, /, //, %	Soldan sağa doğru
	+, -	Soldan sağa doğru
	<, <=, >, >=, !=, ==	Karşılaştırma operatörleri
	not	Mantıksal DEĞİL
	and	Mantıksal VE
	or	Mantıksal VEYA
En Düşük	=, +=, *=, …	Atama operatörleri

4. KOŞUL İFADELERİ

Normal bir programın işleyişi, ilk satırdan son satıra doğru sırası ile adım adım gerçekleşir. Fakat bu işleyiş sırası, bir koşula bağlı olarak değiştirilebilir. Programın işleyiş sırasının bir koşula bağlı olarak değiştirilmesi işlemine şartlı dallanma (branching) adı verilir ve dallanma işlemini gerçekleştiren komutlara da karar komutları denir [1].

Karar yapılarında bir şarta (koşula) bağlı olarak iki ya da daha fazla seçenekten birini seçme işlemi gerçekleştirilmektedir. Program şarta bağlı olarak hangi işlemi yapacağına karar verir. Python'da *'if', 'if else', 'if-elif-else'* olmak üzere üç adet seçme işlemini gerçekleştiren yapı vardır [1].

4.1 Tek Seçimli Yapı (if)

İngilizce bir kelime olan '*if*', Türkçede 'eğer' anlamına gelmektedir. Anlamından da anlaşılacağı üzere bu kelime bir koşul bildirmektedir. Bu yapıda bazı koşullar doğruysa, bazı eylemler gerçekleştirilir. Doğru değil ise, isteğe bağlı olarak bunun yerine başka eylemler gerçekleştirilebilir.

Bu yapıya günlük hayattan pek çok örnek verilebilir:

"Eğer ders ortalaman 50'nin üzerinde ise dersi geçtin."

"Eğer yemek pişti ise fırını kapat."

"if" yapısı basitçe şu şekilde ifade edilebilir:

```
if koşul:
    işlem
işlem2
```

Tek seçimli if yapısında yukarıda olduğu gibi verilen koşul doğru (True) ise 'işlem' bölgesi çalışır. Verilen koşul doğru değil ise program 'işlem' kısmını atlayarak 'işlem2' kısmından çalışmaya devam eder.

Koşul kısmında birden fazla değerin karşılaştırılması gerektiği durumlarda mantıksal operatörler (and, or) yardımıyla koşulları birbirlerine bağlanabilir.

4.1.1 Kod Bloğu: Girinti Mekanizması (Indentation)

Programlama dillerinde kod bloklarını belirlemek için süslü parantezler {…} (Örneğin; C/C++, Java) ya da begin … end (Örneğin; Delphi) sözcükleri kullanılırken Python'da ise girinti mekanizmasından yararlanılır [1].

Girinti, bir kod parçasının başka bir kod parçasıyla nasıl ilişkilendirilmesi gerektiğini belirlemek için kullanılır [2]. Python'da dikey olarak aynı hizada başlayan ardışık kod satırları bir blok kabul edilir ve her bir blok girintisi için 4 boşluk bırakılır.

Örneğin; girilen bir sayının pozitif veya negatif olduğunu gösteren program oluşturmak istenirse algoritma olarak gösterimi:

1. Başlangıç
2. Sayı girişi (a=8)
3. Eğer (a>0) ise "Pozitiftir" yazıdır.
4. Eğer (a<0) ise "Negatiftir" yazdır.
5. Dur

Python kodu olarak gösterimi ise:

```
a=int(input('Bir sayi giriniz:'))
if(a>0):
    print("Pozitiftir.")
if(a<0):
    print("Negatiftir.")
print("Programın sonu")
```

şeklindedir. Koda girdiğimiz sayının 8 olması durumunda (a=8) kodun çıktısı:

```
Pozitiftir.
Programın sonu
```

şeklinde olacaktır. Program, diğer kod bloklarını verilen şartı sağlamadıklarından dolayı çalıştırmayacaktır. Oluşturduğumuz bu koda a=0 girdisini yazmamız durumunda ise çıktı

```
Programın sonu
```

şeklinde olacaktır.

Oluşturulan girintilerin ‘4 boşluk’tan farklı bir şekilde yazımı durumunda (Örneğin; 3 boşluk), program “Bad Indentation” hatası verir.

4.2 Çift Seçimli Yapı (if: else:)

Bir *“if”* cümlesinin ardından isteğe bağlı olarak bir *“else”* ifadesi gelebilir. Else cümlesi yalnızca “if” ifadesinin koşulu False olduğunda çalıştırılır [5].

Bu yapının kullanımına günlük hayattan örnek verecek olursak:

“Eğer ders ortalaman 50’nin üzerinde ise dersi geçtin, değilse kaldın.”

“Eğer fırının derecesi 100 ise fırını kapat, değilse kapatma.”

if-else yapısının kod üzerinde gösterimini inceleyelim.

```
if koşul:
    İşlem1
else:
    İşlem2
```

Burada verilen koşul doğru (True) ise if bloğundaki işlem yani işlem1 gerçekleşir. Verilen koşulun yanlış (False) olması durumunda ise else bloğundaki işlem yani işlem2 gerçekleşecektir.

4.3 Çok Seçimli Yapı (if: elif: else:)

“if” veya *“else”* cümlelerinden yalnızca biri uygulanacak olsa da, birçok olası cümlecikten birinin yürütülmesinin istenildiği bir durum olabilir. *“elif”* ifadesi, her zaman bir *“if”* veya başka bir *“elif”* ifadesini izleyen bir *"else if"* ifadesidir. Yalnızca önceki koşulların tümü Yanlış ise kontrol edilen başka bir koşul sağlar [5].

“if-elif-else” kullanımı inceleyelim:

```
if koşul 1:
    işlem 1
elif koşul 2:
    işlem 2
elif koşul 3:
    işlem 3
…
else:
   işlem N
```

Program yukarıdan aşağıya doğru koşulları kontrol eder. İlk doğru (True) koşulun tespit edilmesi durumunda program o işlem bloğunu çalıştırır.

5. DÖNGÜLER

Döngüler, tekrar eden işlemleri gerçekleştirmek için tasarlanmış yapılardır. Bu nedenle döngü yapıları, farklı kaynaklarda tekrarlı yapılar olarak da adlandırılmaktadır. Programcı, yazdığı programın bazı kod satırlarını tekrarlı olarak çalıştırma ihtiyacı duyduğunda döngü yapılarını kullanır [1].

Python'da döngü yapıları ve komutları iki başlıkta incelenebilir:

- while döngüleri
- for döngüleri

5.1 while Döngüsü

while döngüsü hemen hemen tüm programlama dillerinde mevcuttur ve verilen koşul (ifade) doğru olduğu sürece bir veya daha fazla kod ifadesini yinelemek (veya tekrarlamak) için kullanılır [2].

Bu döngü yapısı genellikle, kod bloğunun tekrar etmesi gereken sayı bilinmediğinde kullanılır. Örneğin; istenilen çözüm bulunana kadar veya kullanıcı belirli bir değer girene kadar döngü çalıştırmak istenebilir. Bu gibi durumlarda while döngüsü uygun bir seçenek olacaktır.

while yapısını basitçe inceleyelim:

```
while koşul:
    blok işlemleri
…
```

Yapıda verilen koşul ifadesi doğru (True) olduğu sürece döngü çalışacaktır ve içerisindeki blok işlemleri tekrarlanacaktır. Koşul kontrolü döngünün başında yapılmaktadır ve koşulun yanlış (False) olması durumunda döngü sonlanacaktır.

Örneğin; while döngüsü kullanarak, girilen sayının faktöriyelini hesaplayan bir program oluşturalım:

```
f=1
n=int(input("Bir sayi giriniz: "))
i=n
```

```
while i>=1:
    f=f*i
    i=i-1
print(n,”!=”,f)
```

oluşturulan koddaki döngü, “i” değişkeni 1 den büyük olduğu sürece çalışacaktır. Döngü bloğu içerisinde “i” değişkeni her seferinde 1 azaltılmaktadır.

Döngüleri oluştururken döngü içerisindeki kod satırlarının hep aynı hizada aynı girintiye (4 boşluk) sahip olmalarına dikkat edilmelidir. Bir kod bloğu içerisinde başka kod blokları da yer alabilir fakat her yeni kod bloğu 4 karakter daha ötelenerek hizalanmalıdır.

5.2 For Döngüsü

For döngüsü, tekrar sayısının baştan belli olduğu, ardışık eleman listesinin sırayla işleme konulduğu döngü yapısıdır. Örneğin; 1’den 10’a kadar olan sayıları toplayan bir programda, 9 kez tekrarlama yaptırılıyorsa bu tekrarlama sayısının başlangıçta belli olduğu anlamına gelir ki; bu işlem for döngüleri ile yapılabilir [1].

For döngüleri liste/dizi gibi sıralı elemanlar üzerinde işlem yaparlar. Bir for döngüsünün yapısı while döngüsüne benzer, çünkü bir for ifadesi ve bir döngü gövdesi vardır:

```
friends = ['Joseph', 'Glenn', 'Sally']
for friend in friends:
    print('Happy New Year:', friend)
print('Done!')
```

Python terimlerinde, friends değişkeni üç adet string’den oluşan bir listedir. Kodda yer alan for döngüsü listede geçen ve listedeki üç string’in her biri için gövdeyi bir kez çalıştırır ve çıktı şu şekilde olur:

```
Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally
Done!
```

Çalıştırılan kodda friends adlı kümedeki her arkadaş için for döngüsünün gövdesindeki ifadeler bir kez çalıştırılmıştır diyebiliriz. Kodda yer alan “friend” yineleme değişkenidir. “friend” değişkeni, döngünün her yinelemesi için değişir ve for döngüsünün ne zaman tamamlanacağını kontrol eder. Yineleme değişkeni “friends” değişkeninde depolanan stringleri sırasıyla kullanır [3].

5.2.1 range() Fonksiyonu

for döngü yapısında sıklıkla kullanılan ve isteğe bağlı olarak üç parametre alan bir fonksiyondur. Kullanım şekli:

```
range(ilk, son, adım)
```

şeklindedir. range fonksiyonunun ilk parametresi belirtilmezse 0, adım değeri belirtilmezse 1 kabul edilir. Örneğin;
`range(5)`: 0’dan 5’e kadar (5 hariç) sayılar,
`range(3,15)`: 3’den 15’e kadar (3 dahil, 15 hariç) sayılar,
`range(3,15,2)`: 3’den 15’e kadar (15 hariç) sayıları 2’şer arttırarak işlem gerçekleştirir [1].

“reversed()” fonksiyonu ise bir döngü içerisinde yer alan sayıların ters sırada işlem görmelerini sağlar. Örneğin;
`reversed(range(3,15,2))` : 15’den 3’e kadar sayıları 2’şer azaltarak ’13,11,9,7,5,3’ şeklinde listeler.

5.3 Break ve Continue Komutları

”break” ve “continue” komutları döngülerde ek kontrol sağlar. break komutu, bir koşula bağlı olarak döngü değişkeninin son değerine ulaşmadan döngüyü terk etmesini (döngüyü kırmasını) sağlar. continue komutu ise, döngü içerisinde o anki işlemi atlamamızı/göz ardı etmemizi sağlar fakat döngüden bir çıkış sağlamaz [1].
Daha önce oluşturmuş olduğumuz while döngüsünü tekrar inceleyelim ve şu şekilde düzenleyelim:

```
f=1
n=int(input("Bir sayi giriniz: "))
i=n
while i>=1:
    f=f*i
    print("Program calisiyor…")
print(n,"!=",f)
```

kod çalıştırıldığında istenilen girdinin n=5 (veya başka bir pozitif tamsayı) girilmesi durumunda ekrana sürekli "Program calisiyor…" yazısını yazar. Yaptığımız düzenlemede i değişkenin azaltıldığı satırı silmiş olduk ve i değişkeni daima başlangıçta girdiğimiz sayı değeri ile aynı kalmış oldu. Böylece döngüdeki koşul daima doğru (True) değerinde oldu ve sonsuz bir döngü oluştu. Peki bir sonsuz döngünün istenilen anda durdurulması nasıl sağlanabilir?

Sonsuz döngülerden çıkış, döngü kırılarak break komutu ile gerçekleştirilir. Break komutu bir koşula bağlı olarak while ve for döngülerinden çıkmak için yani döngüleri sonlandırmak için kullanılır. Program akışını döngünün dışındaki ilk komuta atlatır [1].

İncelediğimiz örneğin n=5 değerinin faktöriyelinin hesaplandığı anda döngüden çıkmasını istersek:

```
f=1
n=int(input("Bir sayi giriniz: "))
i=n
while i>=1:
    f=f*i
    print("Program calisiyor…")
    if(f==125):
        break
print(n,"!=",f)
```

break komutunu ekleyerek döngünün istenilen durumda kırılması sağlanabilir. Programa "f değişkeninin 125 olması durumunda döngüyü kır" komutu verilmiş oldu. Bu durumda n=5 girilmesi durumunda oluşan çıktı:

```
Bir sayi giriniz: 5
Program çalışıyor
Program çalışıyor
Program çalışıyor
5!= 125
```

şeklinde olur.

continue komutu, döngü içerisinde o anki işlemin atlanmasını (göz ardı edilmesini) ve bir sonraki işlemden devam edilmesini sağlar [1].
continue komutunun kullanımını bir örnek program üzerinden inceleyelim:

```
s=0
while s<7
    s=s+1
    if(s==2 or s==5):
        continue
    print(s)
print("Donguden çikildi")
```

programda normalde 1'den 7'ye kadar olan sayılar ekranda görülecektir. Fakat s=2 ve s=5 değerlerini aldığında *'continue'* ifadesi programın döngüdeki bir satır atlamasını sağlar ve programın ekran çıktısında 2 ve 5 rakamları atlanmış olur. Programın ekran çıktısı aşağıdaki gibi gözlemlenir:

```
1
3
4
6
7
Donguden çikildi
```

6. FONKSİYONLAR

Önceki bölümlerdeki print(), input(), type() gibi fonksiyonlardan bahsedildi. Python'da bunlar gibi birçok önceden tanımlanmış fonksiyonlar bulunmaktadır. Ancak onlar haricinde de isteğe göre fonksiyonlar oluşturulabilir. Bir fonksiyon, bir program içindeki bir mini program gibidir.
Peki bu fonksiyonlar niçin kullanılır?
Bir programın içinde aynı işi gören bir grup kod satırının programın çeşitli yerlerinde tekrar ve tekrar yazılıp kullanılması yerine, bu kod satırlarını bir defa yazılıp sürekli çağırılabilir. Bunun için bu kod grubunun bir fonksiyon başlığı/ismi altında tanımlanması yeterlidir. Fonksiyonlar ile hem kodlama tekrar önlenmiş hem de programın anlaşılabilirliği artmış olur[1].
Fonksiyonlar kendi içlerinde;

- Parametreli (dışarıdan değer alan) veya parametresiz (dışarıdan değer almayan),
- Geriye (Çağırıldığı yere) değer döndüren veya değer döndürmeyen fonksiyonlar

şeklinde sınıflandırılabilir.

6.1 Fonksiyon Tanımlama

Bir fonksiyon tanımı, yeni bir fonksiyonun adını ve fonksiyon çağrıldığında yürütülen ifadelerin sırasını belirtir. Bir fonksiyonu tanımlandığında, fonksiyon program boyunca tekrar tekrar kullanılabilir [3].
Python'da fonksiyonlar, tipik olarak belirli bir görevi gerçekleştiren ve bir dizi parametre alabilen veya almayabilen veya bir değer döndürebilen ilişkili ifadeler gruplarıdır [2].

Fonksiyon tanımlamak için *"def"* anahtar kelimesi kullanılır. Python'da fonksiyon tanımı şu şekilde yapılır:

```
def fonksiyon_adi():
    fonksiyon gövdesi
```

Tanımlanan bir fonksiyon program içerisinde sadece isim, "fonksiyon_adi()" gibi belirtilerek çağırılır. Örneğin ekranda 'Merhaba Dunya' çıktısı oluşturacak bir fonksiyon tanımlanmak istenirse:

```
def selamla():
    print('Merhaba Dunya')
```

şeklinde *selamla()* adlı bir fonksiyon oluşturulabilir. Bu fonksiyonun program içerisinde çağırılması içinse:

```
selamla()
```

satırı kullanılmalıdır. Tanımlanan *selamla()* fonksiyonu herhangi bir parametre almamıştır. Dolayısıyla parametresiz bir fonksiyondur.

Şimdi oluşturulan fonksiyonu isme göre olarak selamlayacak bir yapıya yani dönüştürelim. Bunun için parametrelerden yararlanılmalıdır.

Fonksiyona parametre olarak nasıl bir isim verildiğinin önemi yoktur. Parantez içine parametre olarak istenilen kelime yazılabilir. Önemli olan, parantez içinde fonksiyonun kaç parametre alacağını gösteren bir işaret olmasıdır. Fakat parametrenin görevine uygun bir isim verilmesi fonksiyonun daha okunaklı olmasını sağlayacaktır [4].

```
def selamla(isim):
    print('Merhaba',isim)
```

oluşturulan fonksiyon *isim* adında bir parametre almaktadır. Bu fonksiyon program içerisinde çağırılmak istenirse parantezlerin içerisine fonksiyona gönderilmek istenen değer veya değişken yazılabilir.

```
selamla(“Aslihan”)
```

Burada fonksiyon çağırılırken kullanılan *“Aslihan”* değerine argüman adı verilir. Fonksiyon bu şekilde çağırıldığında oluşan çıktı aşağıdaki gibi olacaktır.

```
Merhaba Aslihan
```

6.2 Fonksiyonlarda Değer Döndürme

Daha önce öğrenmiş olduğumuz input() fonksiyonu kullanım olarak girilen değere program içerisinde erişmemize olanak sağlıyordu. Diğer bir deyişle;

```
ad=input(“isminizi girin:”)
```

komutuna isim girildiğinde programda girilen isim *“ad”* değişkeni içerisinde tutulmaktadır. Burada input() fonksiyonu girilen değeri döndürmüş olur.

Genel olarak, bir fonksiyon çağrısının değerlendirdiği değere fonksiyonun dönüş (return) değeri denir [5]. “return” komutu fonksiyonun geriye döndüreceği değeri üreten komuttur. Örneğin;

```
def selamla():
    isim=input("Merhaba, adın ne?")
    return isim
print("Nasilsin",selamla(),"?")
```

programında *selamla()* fonksiyonundan, fonksiyonun çalışması sırasında girilen *“isim”* değerini döndürmüş olduk. Döndürülen *isim* değeri fonksiyonun dışındaki bir satırda kullanılabilir olmuştur. Fonksiyonu çalıştırdığımızda oluşacak çıktı:

```
Merhaba, adın ne?Aslihan
Nasilsin Aslihan ?
```

olarak görülecektir. Burada input() fonksiyonu “Aslihan” değerini döndürmüş ve *isim* değişkenine aktarmıştır. Ardından, oluşturulan *selamla()* fonksiyonu *isim* değişenindeki “Aslihan” değerini fonksiyonun çağırıldığı yere döndürmüştür ve fonksiyondan gelen değerin, fonksiyonun dışında bulunan print() fonksiyonu içerisinde kullanılması sağlanmıştır.

Şimdi hem parametreli hem de değer döndüren bir fonksiyon örneğini inceleyelim.

```
def Toplama(x,y):
    toplam=x+y
    return toplam
print(Toplama(3,9))
```

Ekran çıktısı:

```
12
```

Verilen örnekte *"Toplama"* adlı fonksiyon iki adet parametre almaktadır. Alınan bu iki parametreyi toplayıp, ardından sonuç değerini döndürmektedir.

6.3 Lambda Fonksiyonu

'lambda' anahtar kelimesi kullanılarak tek satırlık kısa fonksiyonlar oluşturulabilir.

```
fonksiyon_ismi= lambda değişkenler: istenilen komut
```

şeklinde bir yapı ile kullanılır.

Daha önce oluşturulan Toplam fonksiyonunu bir de lambda anahtar kelimesini kullanarak oluşturalım:

```
a=3
b=9
Toplama=lambda a,b:a+b
print(Toplama(3,9))
```

Programın ekran çıktısı bir önceki örnekte olduğu gibi yine '12' olacaktır.

6.4 Değişkenlerin Faaliyet Alanları

Bir değişken, programın tamamında ya da sadece belli bir alanında faaliyet gösterebilir, hayatta kalabilir. Faaliyet gösterdiği alan dikkate alınarak değişkenler genel (global) ve yerel (local) değişkenler olmak üzere ikiye ayrılır. Programın tamamında faaliyet gösteren değişkenlere genel (global) değişken, programın sadece belli bir kısmında, fonksiyon veya kod bloğunda faaliyet gösteren değişkenlere yerel (local) değişken denmektedir. [1]

Yerel bir değişken global olarak tanımlanabilir. Bunun için değişkenin başına 'global' anahtar kelimesi getirilir. Örneğin;

```
global s
```

6.5 Rastgele Sayı Üretimi

Aynı girdiler verildiğinde, çoğu bilgisayar programı her seferinde aynı çıktıları üretir, bu nedenle bunların deterministik olduğu söylenir. Aynı hesaplamanın aynı sonucu vermesini beklenildiği için determinizm genellikle iyi bir şeydir. Yine de bazı uygulamalar için bilgisayarın tahmin edilemez olması istenir. Oyunlar bariz bir örnektir, ancak daha fazlası da vardır [3].

Python'da rastgele sayı üretimi için programa random modülü eklenmeli ve bu modülde bulunan random(), randrange() gibi fonksiyonlar kullanılmalıdır. Random modülünü programa dahil etmek için *'import random'* satırı eklenmelidir.

Random modülüne ait bazı fonksiyonlar ve açıklamaları:

- **`random.random()`** : '0.0<=sayi<1.0' değerleri arasında float tipinde rastgele bir sayı üretir.
- **`random.randint(s1,s2)`** : 's1<=sayi<=s2' olacak şekilde int tipinde rastgele bir sayı üretir.
- **`random.randrange(s1,s2,[adım])`** : 's1-s2' değerleri arasında adım miktarına dikkat edilerek float tipinde rastgele bir sayı üretir, adım belirtilmezse 1 kabul edilir.
- **`random.uniform(s1,s2)`** : 's1<=sayi<=s2' olacak şekilde float tipinde rastgele bir sayı üretir.

Random modülünde bulunan fonksiyonların kullanımına dair bir örnek inceleyelim:

```
import random
for i in range(5):
    print(random.randint(1,50),end=' ')
```

Program çalıştırıldığında oluşabilecek örnek ekran çıktısı;

```
8 18 37 8 35
```

şeklinde olabilir. Bu oluşan liste programın her çalıştırılmasında değişecektir.

6.6 Sayı Tahmini Oyunu

Bilgisayar tarafından rastgele belirlenen sayının, kullanıcı tarafından tahmin edildiği bir oyun programı yazalım:

Programda, bilgisayar 1 ile 50 arasında rastgele bir sayı tutsun ve tutulan bu sayının kullanıcı tarafından tahmin edilmesi istensin. Ardından kullanıcının girmiş olduğu sayıya göre "daha büyük veya daha küçük bir sayı giriniz" gibi yönlendirmeler yapılsın. Yapılan tahmin sayısına göre puan hesaplansın ve oyun sonunda tahmin sayısı ile birlikte söylensin.

Bunun için öncelikle rastgele bir sayı oluşturulacağı için random modülü programa dahil edilmelidir. Random modülünde bulunan randint() fonksiyonu yardımıyla 1 ile 50 arasında bir sayı oluşturulup bir değişkene aktarılmalıdır.

```
import random
TutulanSayi=random.randint(1,50)
```

Sonrasında sayı tahminini kontrol edecek ve ilk değeri fonksiyona taşıyacak parametreli bir fonksiyon tanımlanmalıdır. Fonksiyonun içerisinde girilen sayının tutulan sayı ile aynı olup olmadığı değerlendirilmeli ve ona göre bir geri bildirim oluşturulup, farklı değerler olması durumunda yeniden tahminde bulunulması istenmelidir. Bunu sağlamak için döngü kullanılmalıdır.

```
def sayiTahmin(parametre1):
    while True:
        if parametre1==TutulanSayi:
            print("Tebrikler! Doğru bildiniz.")
            break
        elif(parametre1<TutulanSayi):
            print("Üzgünüm yanlış oldu. Daha büyük bir sayı dene!")
            parametre1=int(input("Yeni sayi:"))
        else:
            print("Üzgünüm yanlış oldu. Daha küçük bir sayı dene!")
            parametre1=int(input("Yeni sayi:"))
```

Programa puan hesaplama özelliği ve kaçıncı denemede tahminin doğru bulunduğunu hesaplaması için sayaç değişkeni de eklenmelidir. Bazı gerekli değişikliklerin de yapılması durumunda sayı tahmin oyunu Şekil 6.1.'de görüldüğü üzere hazır olacaktır.

SayiTahminOyunu.py

```
import random

#Tanımlanan fonksiyon
def sayiTahmin(parametre1):
    while True:
        global sayac
        sayac+=1
        puan=100
        if parametre1==TutulanSayi:
            print("Tebrikler!",sayac,". denemede doğru bildiniz.")
            print("Puanınız:",puan-(sayac*10))
            break
        elif(parametre1<TutulanSayi):
            print("Üzgünüm yanlış oldu. Daha büyük bir sayı dene!")
            parametre1=int(input("Yeni sayi:"))
        else:
            print("Üzgünüm yanlış oldu. Daha küçük bir sayı dene!")
            parametre1=int(input("Yeni sayi:"))

#Ana program
TutulanSayi=random.randint(1,50)
sayac=0
print("***Sayi tahmin etme oyununa hoş geldiniz***")
print("1 ile 50 arasında bir sayı tuttuk, bu sayıyı tahmin etmek ister misiniz?")
n=int(input("İlk tahmininizi girin:"))
sayiTahmin(n)

```

Şekil 6.1. Sayı tahmin oyunun kod ile gösterimi

Ana programda tanımlanan *"sayac"* değişkeni fonksiyon içerisinde değişiklik yapılarak kullanılması gerekmektedir. Bu yüzden fonksiyon içerisinde, değişken üzerinde işlem yapmadan önce global olarak tanımlanmalıdır.

Yazılan kod çalıştırıldığında girilen değerlere göre örnek bir ekran çıktısı Şekil 6.2'deki gibi olabilir.

```
In [20]: runfile('C:/Users/aslih/.spyder-py3/temp.py', wdir='C:/Users/aslih/.spyder-
py3')
***Sayi tahmin etme oyununa hoş geldiniz***
1 ile 50 arasında bir sayı tuttuk, bu sayıyı tahmin etmek ister misiniz?

İlk tahmininizi girin:15
Üzgünüm yanlış oldu. Daha büyük bir sayı dene!

Yeni sayi:25
Üzgünüm yanlış oldu. Daha büyük bir sayı dene!

Yeni sayi:40
Üzgünüm yanlış oldu. Daha küçük bir sayı dene!

Yeni sayi:30
Üzgünüm yanlış oldu. Daha büyük bir sayı dene!

Yeni sayi:32
Üzgünüm yanlış oldu. Daha büyük bir sayı dene!

Yeni sayi:34
Üzgünüm yanlış oldu. Daha büyük bir sayı dene!

Yeni sayi:38
Üzgünüm yanlış oldu. Daha büyük bir sayı dene!

Yeni sayi:39
Tebrikler! 8 . denemede doğru bildiniz.
Puanınız: 20
```

Şekil 6.2. Sayı tahmin oyununun ekran çıktısı

7. KARAKTER DİZİLERİ (STRINGS)

Karakter dizileri, adından da anlaşılacağı gibi, karakterlerin bir araya gelmesiyle oluşan bir dizidir. Karakter dizileri tek, çift veya üç tırnak içinde gösterilen, öteki veri tiplerinden de bu tırnaklar aracılığıyla ayırt edilen özel bir veri tipidir. Teknik olarak ifade etmek gerekirse, bir nesneyi *type()* fonksiyonu yardımıyla sorguladığımızda, eğer *<class 'str'>* çıktısı alınıyorsa bu nesne bir karakter dizisidir [4].

Bir string'in elemanlarına erişmek için indisler kullanılır. String değişkeninin isminin sonrasında gelen köşeli parantezlerin içerisindeki tam sayıya indis (index) adı verilir. String'lerde indis numarası 0'dan başlar. Ancak Python dilinde indisler negatif değerler de alabilir ve negatif değerler alması durumunda indis numaraları sondan -1 ile başlar. Örneğin;

```
K="Python"
```

şeklinde tanımlanan bir karakter dizisinin 0. Elemanına, yani *'P'* karakterine K[0] ile veya K[-6] ile erişilebilir. Benzer şekilde *'n'* karakterine ulaşmak için K[5] veya K[-1] kullanılabilir.

7.1 String Dilimleme

Karakter dizilerini parçalara ayırmak (dilimlemek) için ':' operatörü kullanılır. String dilimleme işleminde ':' iki nokta operatörünün kullanım şekli:

```
karakter_dizisi[alınacak_ilk_ögenin_indisi:alınacak_son_ögenin_indis
inin_bir_fazlası]
```

Eğer ilk indis değeri belirtilmezse '0', son indis değeri belirtilmezse 'dizinin toplam uzunluğu' kabul edilir. Adım miktarının varsayılan değeri '1' dir [1]. String dilimlemeyi bir örnek üzerinde basitçe inceleyelim:

```
>>> karakter_dizisi = "Istanbul"
>>> print(karakter_dizisi[0:3])
Ist
```

Yazılan indis değerleri doğrultusunda ekran çıktısında 0,1 ve 2. elemanlar görülür. 3. elemanın dahil olmadığına dikkat edilmelidir. Benzer şekilde aynı karakter dizisi için;

karakter_dizisi[1::2] : 1.indisli elemandan başlayıp ikişer atlayarak dizinin sonuna gider.

Ekran çıktısı: *sabl*

karakter_dizisi[:6:3] : 0.indisli elemandan başlar ve 6.indisli elemana kadar 3'er 3'er ilerler.

Ekran çıktısı: *ia*

Python, bir stringin elemanlarının doğrudan silinmesine izin vermez. Fakat bir string bir bütün olarak silinebilir [1]. Bunun için del fonksiyonu *"del karakter_dizisi"* komutu kullanılabilir. Stringler değişmezdir (immutable), yani mevcut bir dizeyi değiştirilemez. Karakter değiştirmek istendiğinde yapılabilecek en uygun yöntem, orijinalin bir varyasyonu olan yeni bir dize oluşturmaktır [3]:

```
>>> selamla = 'Hello, world!'
>>> yeni_selamla = 'J' + selamla[1:]
>>> print(yeni_selamla)
Jello, world!
```

7.2 String Metotları

Stringler Python nesnelerine bir örnektir. Python, bir nesne için kullanılabilen metotları listeleyen *'dir'* adında bir fonksiyona sahiptir. *'type()'* fonksiyonu bir nesnenin türünü gösterirken *'dir()'* fonksiyonu ise mevcut fonksiyonları gösterir [3].

```
>>>dir(str)
```

Verilen komut karakter dizilerinin bütün metotlarını listeler. Komut çalıştırıldığında 40'ın üzerinde string metodunun olduğu görülür. Bu metotlar içerisinde sıkça kullanılanları Çizelge 7.1.' de verilmiştir.

Python'da bir karakter dizisinin uzunluğunu bulmak için tıpkı C dilindeki *strlen()* fonksiyonuna benzer şekilde *len()* fonksiyonu kullanılmaktadır.

```
>>>selamla='Hello, world!'
>>>print(len(selamla))
13
```

Çizelge 7.1. String işlem metotları

String Metotları	Açıklaması	Örnek Kullanım
S.capitalize()	S'nin ilk karakterini büyük harfe dönüştürür.	print("hello".capitalize) #Hello
S.count("e")	S'nin içerisinde bulunan "e" adetini döndürür.	print("hello".count("e")) #1
S.endswith(t)	S'nin t ile bitip bitmediğini kontrol eder, t ile bitiyorsa True, bitmiyorsa False değerini döndürür.	print("hello".endswith("lo")) #True
S.startswith(t)	S'nin t ile başlayıp başlamadığını kontrol eder, t ile başlıyorsa True, başlamıyorsa False değerini döndürür.	print("hello".startswith("hel")) #True
S.lower()	S'yi küçük harfe dönüştürür.	print("HELLO".lower()) #hello
S.upper()	S'yi büyük harfe dönüştürür.	print("hello".upper()) #HELLO
S.replace("a","b")	S'in içindeki tüm a karakterlerini, b karakterine dönüştürür.	print("hello". replace("e","a")) #hallo
S.split("ayraç") veya S.split()	S içerisinde belirtilen ayraç karakteri veya boşluklar dikkate alınarak metin parçalanır ve listeye dönüştürülür.	print("hello beautiful world". split()) #['hello', ' beautiful', 'world']
S.islower()	S tamamen küçük harf ise True, değilse False değerini üretir.	print("hello".islower()) #True
S.isupper()	S tamamen büyük harf ise True, değilse False değerini üretir.	print("hello".isupper()) #False

8. VERİ YAPILARI

Algoritmalar tarafından işlenen en temel elemanlara (sayısal bilgiler, metinsel bilgiler, resimler, sesler vb.) *veri* adı verilir. Bir algoritmanın etkin, anlaşılır ve doğru olabilmesi için algoritmanın işleyeceği verilerin düzenlenmesi, işlenmesi gerekir [1].

Verilerin düzenlenme biçimini belirleyen yapıtaşlarına veri yapıları denir. Python programlama dili dört temel veri yapısına sahiptir. Bunlar liste (list), demet (tuple), küme (set) ve sözlük (dict) olarak isimlendirilir.

Kümeler, listeler ve sözlükler değiştirilebilir (mutable) veri tipleri iken demetler ve karakter dizileri (strings) değiştirilemez (immutable) veri tipinde yapılardır [1].

8.1 Liste (List) Yapısı

Python'da listeler, klasik anlamda dizilerin (arrays) yerini alan yapılardır. Fakat çok daha işlevseldir. Klasik dizi yapılarından farklı olarak listeler aynı tip elemanlardan oluşmak zorunda değildir [1].

Listeler, sıralı ve değiştirilebilir (mutable) nesnelerin bir koleksiyonunu tutar, indekslenir ve yinelenen üyelere izin verir [2].

Köşeli parantezler arasına yazılan sıralı elemanlar dizisi bir listedir ve veri tipi <list> dir. Bir liste iç içe listelerden de oluşabilir. Aşağıda çeşitli liste örnekleri verilmiştir:

- **L=['P','Y','T','H','O','N']** : 5 elemanlı bir karakter listesi
- **L= ["Aslihan", "Gizem", "Ahmet", "Emre"]** : 4 elemanlı bir string listesi
- **L= [1, 3, 5, 7, 9, 11]** : 6 elemanlı bir tam sayı listesi
- **L= []** : Boş bir liste
- **L= ['A', 0.2, [1,2,3], "Fatma", 12]** : Farklı tipteki verilerden oluşan bir liste

Bir Python programında "renk" adında bir string listesi oluşturulsun ve içerisine çeşitli elemanlar yerleştirilsin.

```
renk= ["mor", "mavi", "yeşil", "sari"]
```

Python kodu renk[0]'ı *"mor"* olarak, renk[1]'i ise *"mavi"* olarak değerlendirir ve değerlendirme bu şekilde devam eder. Listelerde de karakter dizilerinde olduğu gibi indis numaraları 0'dan başlayabildiği gibi sondan -1'den de başlayabilir. Örneğin belirtilen renk

listesinde, renk[-1] *"sari"* olarak değerlendirilir. Bir listenin elemanlarına indis numaraları ile erişilip elemanlarda değişiklikler yapılabilir.

Listeler karakter dizileri ile aynı dilimleme (: operatörü) işlemlerini destekler [1]. Ayrıca çeşitli liste metotları kullanılarak liste üzerinde her türlü işlem gerçekleştirilebilir. Bir listeye ekleme yapmak için append() metodu kullanılır. Bu metodun kullanım yapısı *liste_adi.append(nesne)* şeklindedir.

```
L=[1, 3, 5, 7]
L.append(9)
```

Verilen kod çalıştırıldığında L isimli listeye 9 elemanı eklenecek ve liste L=[1, 3, 5, 7, 9] haline gelecektir. Örneklerden de görüldüğü gibi liste oluşturmak için öğeleri belirleyip bunların köşeli parantezler içine alınması yeterli oluyor. Bu yöntemin dışında, liste oluşturmanın bir yöntemi daha bulunur [4]. Karakter dizilerini anlatılırken *split()* adlı bir metottan söz edilmişti. Bu metot karakter dizilerinin belli bir ölçüte göre bölünmesini ve liste oluşturulmasını sağlıyor. Ancak *split()* metodunun bir karakter dizisini bölüp bir liste oluşturabilmesi için karakter dizisinin belli bir ölçüte göre bölünebilir durumda olması gerekiyor. Bu şekilde bölünebilir durumda olmayan karakter dizilerini listeye çevirmek için *list()* adlı bir fonksiyon kullanılabilir. Örneğin;

```
karakter_dizi="abcdefgh"
karakter_liste=list(karakter_dizi)
print(karakter_liste)
```

Verilen kodda karakter_dizi adlı stringin karakterlerinden oluşan bir liste elde edilir ve programın ekran çıktısı şu şekilde olur:

```
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

"list()" fonksiyonu *li = list()* şeklinde boş bir liste oluşturmak için de kullanılabilir.

8.2 Demet (Tuple) Yapısı

Demet (tuple) listeye oldukça benzeyen bir değerler dizisidir. Bir demette saklanan değerler herhangi bir veri tipinde olabilir ve indisleri tamsayılardan oluşur. Liste ve demet yapısı arasındaki önemli fark, demetlerin değişmez (immutable) olmasıdır [3]. Demet veri yapısında elemanlar virgüller ile ayrılır. Zorunlu olmasa da, Python kodunda demetlerin rahatça ayırt edilebilmeleri için demetler parantez içerisinde kullanılır.

```
>>> t = ('a', 'b', 'c', 'd', 'e')
```

Demetlerin veri tipi <tuple> dır ve demetler listelere göre daha hızlı çalışırlar. Demetlerin değişmez yapısı sayesinde demet elemanları sözlük (dict) veri yapısında anahtar (key) olarak kullanılabilirken listeler kullanılamaz.

8.3 Küme (Set) Yapısı

Küme (Set), sırasız tekil elemanlar topluluğudur. Küme elemanları süslü parantezler {} arasına yazılır. Matematiksel işlemleri yanında basit ekleme ve silme yöntemleri vardır [1]. Kümelerde elemanlar sıralı olmak zorunda değildir ve küme elemanlarına erişim için indis kullanılmaz. Bir küme içerisinde tekrarlı (aynı) elemanlar bulunamaz. Bir çift sayılar kümesi için şu örnek verilebilir:

```
>>> S = {0, 2, 4, 6, 8}
```

Kümeler *set* kelimesi kullanılarak da tanımlanabilirler. Örneğin; *A=set("abracadabra")* şeklinde bir A kümesi tanımlandığında A kümesinin elemanları *A={a,b,r,c,d}* şeklinde olacaktır.

Liste ve demet yapılarında olduğu gibi küme yapılarında da yeni elemanlar eklenebilir. Fakat sadece elemanları doğrudan değiştirilemeyen veri tipleri (tuple, str, int, float) kümeye eklenebilir [1].

8.4 Sözlük (Dictionary) Yapısı

Sözlük, bir dizi indis (anahtar olarak kullanılır) ile bir dizi değer arasındaki bir eşleşme olarak düşünülebilir. Her anahtar bir değerle eşleşir. Bir anahtar ve bir değerin ilişkilendirilmesine anahtar/değer çifti (key/value pair) veya bazen bir eleman denir [3]. String, liste ve demet yapılarında elemanlara erişim için tamsayı indisler kullanılırken, sözlüklerde anahtarlar kullanılır ve her bir anahtar değeri tekil (unique) olmalıdır. Sözlükler genellikle {anahtar: değeri, …} şeklinde kullanılır fakat sözlük yapısı farklı şekillerde de tanımlanabilir. Aşağıda verilen sözlük tanımlamaları eşdeğerdir:

```
>>>a= dict(bir=1, iki=2, uc=3)
>>>b= {‘bir’: 1, ‘iki’: 2, ‘uc’: 3}
>>>c= dict([(‘iki’, 2), (‘bir’, 1), (‘uc’, 3)])
>>>d=dict({‘uc’: 3, ‘bir’: 1 , ‘iki’: 2})
```

‘dict’ adlı fonksiyon D=dict() şeklinde kullanıldığında eleman içermeyen, D adında yeni bir sözlük oluşturur. Bu fonksiyon yerleşik bir fonksiyon adı olduğu için, onu bir değişken adı olarak kullanmaktan kaçınılmalıdır. Tanımlanmış bir sözlüğe eleman eklemek için D[key]=[değeri] şeklinde bir komut kullanılmalıdır.

```
a={'iki':2, 'uc':3}
a['bir']=1
print(a)
```

Verilen kodun çalıştırılmasıyla oluşacak ekran çıktısı aşağıda verilmiştir.

```
{'iki': 2, 'uc': 3, 'bir': 1}
```

SONUÇLAR

Bu tezde Python Programlama Dilinin anlatımı ele alınmıştır.

Birinci bölümde Python Programlama Dili hakkında genel bilgilere yer verilmiştir. İkinci bölümde Python Programlama Dilinin veri yapısı ve yazım kuralları verilmiştir. Üçüncü bölümde ise tüm programlama dillerinde benzer özellikler gösteren operatörler incelenmiştir. Dördüncü bölümde kontrol deyimleri ayrıca beşinci bölümde de döngü deyimleri incelenmiş ver örneklendirilmiştir. Altıncı bölümde programlama dillerinin olmazsa olmazı alt programlar yani fonksiyonlar ele alınmıştır. Buna ek olarak buraya kadar anlatılan konuların anlaşılmasını kolaylaştıracak "Sayı tahmini" uygulaması verilmiştir. Yedinci bölümde karakter dizileri örnekleriyle incelenmiştir. Son bölümde ise programlama dillerinde diziler olarak ele alınan veri yapıları incelenip, örneklendirilmiştir.

Bu programlama dili, açık kaynaklara sahip ve çok geniş kütüphanelere sahip olduğundan oldukça zengin bir dildir. Ayrıca web uygulamalarında da kullanılmaktadır. En önemli özelliği ise veri bilimi (data science) ve yapay zeka alanlarında kullanılmasıdır. Bu özellikler sayesinde gün geçtikçe yaygınlaşmaktadır.

KAYNAKÇA

[1] Çobanoğlu, B, *"Herkes için Python"*, (Dr. Günhan Bayrak), 2.Baskı, Pusula, İstanbul, 2019.

[2] Hunt, J, "*A Beginners Guide to Python 3 Programming*", (Ian Mackie), Springer, Cham, 2020.

[3] Severance, C, *"Python for Everybody"*, (Elliott Hauser, Sue Blumenberg), Ann Arbor, 2013.

[4] www.python-istihza.yazbel.com

[5] www.automatetheboringstuff.com

[6] www.w3schools.com

ÖZGEÇMİŞ

Ad Soyad: Aslıhan Nur Pehlivan

Doğum Tarihi: 24/07/1997

Doğum Yeri: İstanbul

Lise: Fenerbahçe Anadolu Lisesi (2012-2015)

Uğur Temel Lisesi (2015-2016)

Üniversite: Yıldız Teknik Üniversitesi Fen- Edebiyat Fakültesi – Matematik Bölümü (2016-)

Yıldız Teknik Üniversitesi Kimya-Metalurji Fakültesi – Matematik Mühendisliği Bölümü (2018-)

E-mail: aslihan.nur@hotmail.com