# Programlama Dilleri Laboratuvar Föyü
# Python Programlama Dili 3
# Liste Üreteçleri, Fonksiyonlar, Hata Yakalama ve Sınıflar

Şevket Umut ÇAKIR

## 1 Giriş

Bu deneyde Python dilindeki liste üreteçleri(list comprehensions), fonksiyonlar, hata yakalama ve sınıflar incelenecektir. Dilin bütün özellikleri üç haftalık süreçte anlatılamamaktadır. Verilen bilgiler giriş seviyesinde olup öğrencinin araştırmasına açıktır.

## 2 Konu Anlatımı ve Deney Hazırlığı

### 2.1 Liste Üreteçleri(List Comprehensions)

Liste üreteçleri listeleri oluşturmak için özlü bir yol sunarlar. Genel kullanımı sayılabilir nesneler ve veri yapılarını kullanarak ve bu türlerin elemanları üzerinde çeşitli işlemler gerçekleştirilerek başka sayılabilir türler elde etmek şeklindedir[1].

Yazım biçimi `yeni_liste = [ifade(i) for i in eski_liste if filtre(i)]` biçimindedir. Birden ona kadar olan sayıların karelerinin toplamı aşağıdaki farklı şekillerde yapılabilir. Bunlardan en kısası liste üreteçleridir.

```
# 1. yol: for döngüsü
kareler = []
for i in range(10):
    kareler.append(i**2)

# 2. yol: yüksek dereceden fonksiyonlar
kareler = list(map(lambda x: x**2, range(10)))

# 3. yol: liste üreteçleri
kareler = [x**2 for x in range(10)]
```

Liste üreteçleri tek bir aralık üzerinde kullanılabildiği gibi iç içe de kullanılabilir. Aşağıda liste üreteçlerinin kullanımı ile ilgili örnekler bulunmaktadır.

```
#pozitif olanları alma
print([a for a in [-8, -4, -2, 0, 2, 4, 8] if a>0])
vec = [[1,2,3], [4,5,6], [7,8,9]]
#listeleri düzleştirme(flatten)
print([num for elem in vec for num in elem])
# 100'den küçük asal sayılar
asal = [p for p in range(2,100) if all(p%y!=0 for y in
↪  range(2,p))]
print(asal)
# 1000'den küçük mükemmel sayılar
mukemmel = [p for p in range(2,1000) if sum([d for d in
↪  range(1,p) if p%d==0])==p]
print(mukemmel)
# liste içinde tuple üretme
print([(x, "2'nin katı" if x%2==0 else "2'nin katı değil") for x
↪  in range(20)])
t=(x*2 for x in range(10)) # üreteç nesnesi
print(tuple(t)) # tuple üretme, üretildikten sonra kaybolur
t=(x**2 for x in range(10))
print(list(t)) # liste üretme
```

## 2.2 Fonksiyonlar

Fonksiyonlar belirlenen işlevleri gerçekleştiren kod parçacıklarıdır. Fonksiyonları tanımlamak için `def` anahtar kelimesi, onu takip eden fonksiyon adı ve parantez içerisinde virgüllerle ayrılarak yer alan parametreleri yazılır. Aşağıda n sayısından küçük Fibonacci sayılarını ekrana yazdıran bir fonksiyon tanımı ve bu fonksiyonun 100 parametresi ile çağrılması vardır. Fonksiyonları çağırmadığımız sürece içerisindeki kodları çalıştıramayız(`fib(100)` gibi).

```
def fib(n): # fonksiyon tanımı
    a, b = 0, 1
    while a < n:
        print(a)
        a, b = b, a+b

fib(100) # fonksiyon çağrısı
```

Fonksiyonlardan geriye `return` ifadesi ile değer döndürülür. Tek bir değer döndürülebileceği gibi virgüllerle ayrılarak birden fazla değer de döndürülebilir.

```
def fonksiyon(sayi): # tanım
    if sayi<1000:
        return True, 'Sayı 1000\'den küçük'
    else:
        return False, 'Sayı 1000\'den büyük'

sonuc, metin = fonksiyon(150) # fonksiyon çağrısı
```

### 2.2.1 Varsayılan Argüman Değerleri

Fonksiyonlar tanımlarırken argümanlara varsayılan değerler verilebilir. Bu sayede kullanıcı ilgili parametre için bir değer girmemişse o argümanın değeri varsayılan olarak belirlenir.

```
def fonksiyon(a1, a2='merhaba', a3='dünya!'):
    print('{} {} {}'.format(a1, a2, a3))

fonksiyon('Python')
fonksiyon('Python', 'hello', 'world!')
fonksiyon('Python', a3='programlama')
```

Python dilinde fonksiyonlara değişken sayıda parametre göndermek mümkündür. Bunun için anahtar kelime argümanları(keyword argument) kullanılır. Anahtar kelime argümanları tek "*" sembolü ile tanımlanan yinelenebilir tipte ve de "**" ile tanımlanan sözlük tipinde olmaktadır. Anahtar kelime argümanları fonksiyon parametrelerinin en sağında(sonunda) yer almalıdır.

```
def fonk(arg1, *args, **kwargs):
    print(arg1)
    for arg in args:
        print(arg)
    for k in kwargs:
        print(k, ':', kwargs[k])

fonk('zorunlu')
fonk('deneme', '1. parametre', '2.', 3.14, yaz='python',
↪  anahtar='AE12FF')
""" Çıktı aşağıdaki şekildedir:
zorunlu
deneme
1. parametre
2.
3.14
yaz : python
anahtar : AE12FF
"""
```

### 2.2.2 Lambda İfadeler ve Yüksek Dereceden Fonksiyonlar

Lambda ifadeler anonim(isimsiz) fonksiyonlar oluşturmak içindir. Bir fonksiyona başka bir fonksiyonu parametre olarak gönderiyorsak ya da bir fonksiyonun geri dönüş değeri olarak alabiliyorsak bu tip fonksiyonlara yüksek dereceden fonksiyon denir. Yüksek dereceli fonksiyonlardan en çok kullanılanlarından bazıları map ve filter fonksiyonlarıdır. map fonksiyonu yinelenebilir bir nesnedeki bütün elemanlara bir fonksiyon uygulayarak geriye bir map nesnesi döndürür ve

bu nesne listeye dönüştürülebilir. filter fonksiyonu ise yinelenebilir nesneleri bir fonksiyon kullanarak filtreler. Aşağıda örnek kullanımları vardır:

```
(lambda n: print(n))(5) #isimsiz fonksiyon oluşturma ve değer gönderme
(lambda x: print('çift' if x%2==0 else 'tek'))(13) # tek yazar

def cift(x):
    return x%2 == 0

print(list(map(cift, range(20)))) # True ve False'lardan oluşan liste
print(list(filter(cift, range(20)))) # 20'den küçük çift sayı listesi

def fonk(f, sayi):# fonksiyona başka fonksiyonu parametre olarak gönderme
    print(f(sayi))

fonk(cift, 17) # False yazar
fonk(lambda n: 'çift' if n%2==0 else 'tek', 20) # çift yazar
```

### 2.2.3 pass ifadesi

Bazı ifadelerin gövdesinin boş olmaması gerekir. Bu durumlarda hiç bir iş yapmayan fakat yazım yanlışlarının önüne geçen pass ifadesi kullanılabilir.

```
while True:
    pass #sonsuz döngü

def bos_metod():
    pass #bir iş yapmaz
```

### 2.2.4 Metod mu, Fonksiyon mu?

Tanımlanan kod parçacığı bir sınıfın öğesi ise metod olarak adlandırılır, herhangi bir sınıftan bağımsız ise fonksiyon adını alır. Sınıf konusu görülene kadar kullanılan bütün bloklar fonksiyondur.

## 2.3 Hata Yakalama

Python dilinde iki tür hatadan söz etmek mümkündür: yazım hataları ve olağan dışı durumlar(istisnalar). Yazım hataları kodun çalışmasını engelleyecek türde hatalardır. Olağan dışı durumlar ise kod çalışırken beklenmedik bir durumun ortaya çıkmasıdır. Hata yakalama işlemleri `try` ve `except` komutları ile gerçekleştirilir.

```
while True print('sonsuz döngü') # yazım hatası
y = 0
x = 1 / y # olağan dışı durum

try:
    x=1 / y
except ZeroDivisionError: # sıfıra bölme hatası oluşursa
```

```
    print('Sıfıra bölme hatası')

while True:
    try:
        x = int(input('Bir sayı girin: '))
        break
    except (ValueError, KeyboardInterrupt): #birden fazla hata
    ↪  algılama
        print('Geçerli bir sayı girmediniz')
    except: # diğer tüm hatalar
        print('Beklenmedik hata')
        raise
```

# 3 Deneyin Uygulanması

Deneyde fonksiyonların tanımlanması ve hata yakalama ile ilgili kodların yazılması istenmektedir.

## 3.1 asal fonksiyonu

asal fonksiyonunda parametre olarak verilen n sayısından küçük asal sayılarının listesini geri döndüren fonksiyon yazmanız beklenmektedir. Fonksiyonun örnek çalışması verilmiştir.

```
>>> asal(10)
[2, 3, 5, 7]
```

## 3.2 mukemmel fonksiyonu

mukemmel fonksiyonu parametre olarak verilen bir sayının mükemmel olup olmadığını test eder. Mükemmelse geriye `True` aksi takdirde `False` döndürür. Çalışma şekli aşağıdaki gibidir.

```
>>> mukemmel(28)
True
>>> mukemmel(30)
False
```

## 3.3 asal liste fonksiyonu

`asal_liste` fonksiyonu değişken sayıda verilen parametre için her bir parametrenin asal olup olmadığını gösteren bir liste döndürür. Örnek kullanımı aşağıdaki gibidir:

```
>>> asal_liste(2,3,5,6)
[True, True, True, False]
>>> asal_liste(97,10,21,23,45,47,49,53)
[True, False, False, True, False, True, False, True]
```

### 3.4 Sayıya Böl Topla

Kullanıcı boş metin girene kadar kullanıcıdan tam sayı alacak olan ve 1000 sayısını alınan sayılara bölerek toplayan kodu yazınız. Kullanıcıdan gelen değerler sadece tam sayılar veya geçerli olmayabilir. Eğer kullanıcı sıfır girmişse toplama herhangi bir değer eklenmeyecektir. Buradaki amaç hatalı durumları yakalayan bir uygulama yazmaktır. Programın örnek girdi için verdiği sonuç aşağıdadır:

```
sayı girin: 3
sayi girin: merhaba
sayı girin: 2.5
sayı girin: 0
sayı girin:
833
```

## Kaynaklar


[1] *The Python Tutorial - Data Structures.* URL: `https://docs.python.org/3/tutorial/datastructures.html` (son erişim: 4.3.2019).

[2] *The Python Tutorial - Errors and Exceptions.* URL: `https://docs.python.org/3/tutorial/errors.html` (son erişim: 5.3.2019).

[3] *The Python Tutorial - More Control Flow Tools.* URL: `https://docs.python.org/3/tutorial/controlflow.html` (son erişim: 3.3.2019).