Functions

May 23, 2022

1 Funksiyalar

1.0.1 Öncəki bəhslərdə qeyd etdiyim kimi şərt və dövr operatorları bir proqramlaşdırma dilinin özəyini təşkil etdiyimi kimi funksiyalar da istənilən dildə əsas mövzulardan biridir.Funksiyaları təsnifatlandırsaq,iki yerə bölə bilərik

1.0.2 1. Built-in Function (Kök Funksiyalar)

1.0.3 2. User-defined Functions (İstifadəçi tərəfindən hazırlanmış Funksiyalar)

Bura qədər istifadə etdiyimiz print() funksiyası list() tuple() funksiyaları dil ilə bərabər gələr kök funksiyalardır.Və növbəti bəhslərdə daha geniş formada digər kök funksiyaları öyrənəcəyik.Bu bəhsdə isə istifadəçi tərəfindən hazırlanmış yəni bizim hazırladığımız funksiyaları nəzərdən keçirəcəyikki,funksiyaların nə olduğuna dair təsəvvürümüz olsun.Riyaziyyatda öyrəndiyimiz funksiyalar,dil daxilində də eyni mənanı kəsb edir.f(n)=n+8 funksiydır.n-ə qiymət verməklə funksiyanı həll edə bilərik.Biz isə bu funksiyanı Python dili ilə ifadə edəcəyik.Bəs funksiyaları niyə öyrənirik? Çünki funksiyalar bizim işimizi asanlaşdırır,təkrar kod nümunələri yazmamağımıza yardım edir.Kod blokları artdıqca başda qeyd etdiyimiz iki nöqtə arasında məsafənin tapılması düsturu növbəti sətirlərdə istifadəsi üçün çətinlik törədəcək və dəyişənləri yenidən qeyd etməyimizə ehtiyac olacaqdır.Bunun alternativ yolu funksiya təyin edib daxilində parametrləri verib düsturu təşkil edirik və istənilən kod bloku daxilində funksiyamızı çağırıb hesabatı apara bilərik

Funksiyaları təyin etməyimizin məqsədi müəyyən tapşırıqları yerinə yetirmək üçün istifadə olunur.Özümüz yazdığımız bütün digər funksiyalar istifadəçi tərəfindən təyin olunan funksiyaların altına düşür. Beləliklə, istifadəçi tərəfindən təyin olunan bir funksiya başqasına kitabxana funksiyası ola bilər. Bir proqramda təkrar kod varsa,Funksiya həmin kodları daxil etmək üçün istifadə oluna bilər və funksiyalar çağırıldığı zaman icra olunur. Böyük layihə üzərində işləyən proqramçılar müxtəlif funksiyaları yerinə yetirərək, iş yükünü bölüşə bilər.

Bir funksiya yalnız çağırıldıqda işləyən kod blokudur. Aşağıda qeyd olunan kök funksiyaların bir qismidir.Və dil ilə bərabər gəlir.Biz də kod bloklarında istifadə edirik.

[1]:
```
from stdlib_list import stdlib_list
```

[2]:
```
libraries = stdlib_list("3.7")
```

[3]:
```
libraries[4:50]
```

[3]:
```
['abc',
 'aifc',
```

```
'argparse',
'array',
'ast',
'asynchat',
'asyncio',
'asyncore',
'atexit',
'audioop',
'base64',
'bdb',
'binascii',
'binhex',
'bisect',
'builtins',
'bz2',
'cProfile',
'calendar',
'cgi',
'cgitb',
'chunk',
'cmath',
'cmd',
'code',
'codecs',
'codeop',
'collections',
'collections.abc',
'colorsys',
'compileall',
'concurrent.futures',
'configparser',
'contextlib',
'contextvars',
'copy',
'copyreg',
'crypt',
'csv',
'ctypes',
'curses',
'curses.ascii',
'curses.panel',
'curses.textpad',
'dataclasses',
'datetime']
```

İndi isə istifadəçi tərəfi hazırlanan funksiyaların yazılış qaydasına baxaq.Biz ilk dərslərdə Python daxilində açar sözlərin olduğunu qeyd etmişdik. gər nəzər yetirsəniz def (define) adlı açar söz var

idi.Funksiyaları təyin etmək üçün bu ifadədən istifadə edəcəyik.Aşağıda ümumi halda funksiyanın yazılış metodikası göstərilmişdir

[4]:
```
print("""


        def funksiya_adi (parametr1,parametr2,......):
            #funksiya bloku
            #docstring
            Kodlar
            #geri dönüş dəyəri


""")
```

```
        def funksiya_adi (parametr1,parametr2,…):
            #funksiya bloku
            #docstring
            Kodlar
            #geri dönüş dəyəri
```

Asta-asta ilk funksiyamızı tərtib edək

[5]:
```
def funksiya():
    print('Salam')
```

shift+enter düyməsini sıxdıq aşağı sətrə xətasız keçid etdik

ilk funksiyamızın adını funksiya olaraq qeyd etdik.tipini sorğuya çəkək

[6]:
```
type(funksiya)
```

[6]:
```
function
```

Və tipini soruşduqda bizə function(funksiya) olduğunu bildirdi.Biz funksiyanı tərtib etdik.Amma icra olunmadı.dərsə girişdə də qeyd etdiyim kimi funksiyalar çağırılmadığı müddətcə icra olunmur.İndi gəlin funksiyamızı çağıraq

[7]:
```
funksiya()
```

```
Salam
```

funksiyamız daxilində print() funksiyasından istifadə edərək ekrana Salam ifadəsini çap etdik.

[8]:
```
def salam():
    """
    funksiya haqqında məlumat
    ekrana salam ifadəsi çap olunur
    """
    print('Salam')
    print('Dünya')
salam()
```

```
Salam
Dünya
```

Yuxarıdakı funksiyamız sadəcə ekrana məlumatları göstərir,heç bir riyazi funksiyası yoxdur.

Funksiyaların təşkilində iki əsas ifadə : Parametr və Argumentlər istifadə olunur.Nümunə ilə bu ifadələrlə tanış olaq

[9]:
```
def salam(param):
    print(param)
```

salam funksiyasına param adlı,parametr əlavə etdik.funksiya blokunda print() funksiyası ilə bu parametrə verilən arqumenti çap etmək istəyirik.Aşağıda funksiyanı çağıraq

[10]:
```
salam()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-10-5e16eb799c18> in <module>
----> 1 salam()

TypeError: salam() missing 1 required positional argument: 'param'
```

Və xəta aldıq.Xətada deyildiyi tələb olunan mövqeydə arqument yoxdur.param - parametr,arqument isə parametrə verilən dəyərdir

[11]:
```
def salam(param):
    print(param)
salam('Arqument')
```

```
Arqument
```

Və ekrana arqument ifadəsi çap olundu.salam funksiyasını çağırdıqda yazdığımız Arqument ifadəsi ilk başda qeyd etdiyimiz param ifadəsinin dəyəridir(Arqumentidir).

Funksiyamıza bəzi əlavələr edək

[12]:
```
def salam(ad):
    print('Salam {}'.format(ad))
```

[13]:
```
salam('Eldar')
```

```
Salam Eldar
```

gər funksiyamıza iki arqument versək necə nəticələnəcək!

[14]:
```
salam('Eldar','Arzu')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-f5f5887a71f3> in <module>
----> 1 salam('Eldar','Arzu')

TypeError: salam() takes 1 positional argument but 2 were given
```

TypeError: salam() takes 1 positional argument but 2 were given və xəta aldıq.Xətada da deyildiyi kimi funksiyamız bir arqument aldığı halda biz iki arqument əlavə etmişik.Zatən başda funksiyanın parametr sayından ala bildiyi arqument sayı məlumdur

İki və daha çox arumentlər

[15]:
```
def funksiya(param1,param2):
    print('{} və {} eyni sinifdə oxuyurlar.'.format(param1,param2))
```

[16]:
```
funksiya('Arzu')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-16-23a0cc47778a> in <module>
----> 1 funksiya('Arzu')

TypeError: funksiya() missing 1 required positional argument: 'param2'
```

xəta aldıq TypeError: funksiya() missing 1 required positional argument: 'param2' iki parametr təyin etdiyimiz halda bir arqument əlavə etdik

[17]:
```
def funksiya(param1,param2):
    print('{} və {} eyni sinifdə oxuyurlar.'.format(param1,param2))
funksiya('Arzu','Ilqar')
```

```
Arzu və Ilqar eyni sinifdə oxuyurlar.
```

[18]:
```
def country(country,city,avenu,home):
    print('country:',country)
    print('city:',city)
    print('avenu:',avenu)
    print('home:',home)
```

```
country('Azerbaijan','Ganja','H.Zardabi','ev/1')
```

```
country: Azerbaijan
city: Ganja
avenu: H.Zardabi
home: ev/1
```

Yuxarıdakı nümunədə isə dörd sayda parametr yazıb,country funksiyasını çağırdıqda isə 4ədəd arqument yazaraq ifadələri ekranda göstərdik

Cüt və tək ədədlərin təyini üçün funksiya tərtib edək

[123]:
```
def Num():

    while True:
        i=input('ədəd yazın(Proqramdan çıxmaq üçün \'q\' düyməsini sıxın.):')
        if i == 'q' or i == 'Q':
            print('Proqramdan çıxılır......')
            break;
        elif int(i)%2==0:
            print('{} ədədi cüt ədəddir'.format(i))
        else:
            print('{} ədədi tək ədəddir'.format(i))

Num()
```

```
ədəd yazın(Proqramdan çıxmaq üçün 'q' düyməsini sıxın.):12
12 ədədi cüt ədəddir
ədəd yazın(Proqramdan çıxmaq üçün 'q' düyməsini sıxın.):5
5 ədədi tək ədəddir
ədəd yazın(Proqramdan çıxmaq üçün 'q' düyməsini sıxın.):7
7 ədədi tək ədəddir
ədəd yazın(Proqramdan çıxmaq üçün 'q' düyməsini sıxın.):q
Proqramdan çıxılır…
```

[124]:
```
Num()
```

```
ədəd yazın(Proqramdan çıxmaq üçün 'q' düyməsini sıxın.):12
12 ədədi cüt ədəddir
ədəd yazın(Proqramdan çıxmaq üçün 'q' düyməsini sıxın.):6
6 ədədi cüt ədəddir
ədəd yazın(Proqramdan çıxmaq üçün 'q' düyməsini sıxın.):9
9 ədədi tək ədəddir
ədəd yazın(Proqramdan çıxmaq üçün 'q' düyməsini sıxın.):Q
Proqramdan çıxılır…
```

[21]:
```
def func():
    while True:
        command = input('əmr daxil edin:[qe]: ')
```

```
        #print(type(command)) -string sətir tipi verilən
        if 'q' in command.lower(): #input funksiyası dəyəri hər zaman sətir␣
 ↪tipində olduğu üçün sətir /tipi verilənlərin metodlarını istifadə edə bilərik
            break
        if command.lower() == 'e':
            print('Xoş gəlmisiniz....')
        else:
            print('Invalid command, try again')
func()
```

```
əmr daxil edin:[qe]: e
Xoş gəlmisiniz…
əmr daxil edin:[qe]: e
Xoş gəlmisiniz…
əmr daxil edin:[qe]: q
```

Funksiyalara əlavə etdiyimiz parametrləri birbaşa arqumentlərlə əlaqələndirə bilərik

[22]:
```
def params(string):
    "Funksiya və parametrlər"
    print (string)
params(string = "Python")
```

```
Python
```

[23]:
```
def params (name,surname):
    """  """
    print(name+' '+surname)
params(name='Nazim',surname='Almammadov')
```

```
Nazim Almammadov
```

[24]:
```
def siyahı(arg):
    """
    Siyahı tərtibatı

    """
    arg = [i for i in range(arg)]
    print(arg)
    print('*'*33)
    for j in arg:
        if j%2 != 0:
            print(j)
siyahı(11)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
*********************************
```

```
1
3
5
7
9
```

Yuxarıdakı funksiya daxilində siyahı tərtib etdik,daha sonra siyahı elementlərini for operatoru vasitəsilə hər dövrdə j dəyişəninə mənimsətdik,dövr təkrarlandıqca növbəti if operatoru ilə j-dəyərinin 2-yə bölünməsindən əldə olunan qalıq 0-dan fərqlidirsə(bərabər deyilsə) ekrana tək ədədləri çap etdiririk

[25]:
```
def funksiya(params = 10):
    print('Parametr:',params)
funksiya()
```

```
Parametr: 10
```

Yuxarıdakı nümunədə parametr olaraq qeyd etdiyimiz params ifadəsinə 10 dəyərini verdik,amma funksiyanı çağırdıqda heç bir arqument əlavə etmədik.Funksiyanı təyin edərkən parametrə 10 dəyərini mənimsətdiyimiz üçün default olaraq params arqumentini qiymətləndirir. gər funksiyanı çağırdığımız zaman arqument versək,funksiya default dəyərini deyil növbəti qeyd olunan dəyərə uyğun davranacaq

[26]:
```
def funksiya(params = 10):
    print('Parametr:',params)
funksiya(30)
```

```
Parametr: 30
```

Və nəticə etibarı ilə ekrana 30 dəyəri çap olundu.

[27]:
```
default dəyərə uyğun növbəti funksiya nümunəsi yazaq
```

```
  File "<ipython-input-27-8bc3cdda3405>", line 1
    default dəyərə uyğun növbəti funksiya nümunəsi yazaq
                  ^
SyntaxError: invalid syntax
```

[28]:
```
def info(name='Empty',surname='Empty',email='Empty',mobile='Empty'):
    print('Name:{} \nSurname:{} \nEmail:{} \nMobile:{}'.
 ↪format(name,surname,email,mobile))
info('Rashad','Garayev')
```

```
Name:Rashad
Surname:Garayev
Email:Empty
Mobile:Empty
```

[29]:
```
info('Rashad','Garayev','example@gmail.com','+99455100')
```

```
Name:Rashad
Surname:Garayev
Email:example@gmail.com
Mobile:+99455100
```

Kod nümunələrində parametrlərin sayı məlum olduğundan biz arqumentləri yaza bilir.Bu parametrlərin sayı limitsiz deyil 256-saya qədər limit təşkil edir.Hər dəfə funksiyamıza parametr adını qeyd etmək yerinə,daha alternetiv yolu mövcuddur.Bu sintaksis *args ifadəsilə yazılır.Nümunələrə nəzər yetirək

1.0.4 *Args ifadəsi

[30]:
```
def adder(x, y, z):
    print("Result:",x + y + z)

adder(4,2,8)
```

```
Result: 14
```

Yuxarıda 3 parametr(x,y,z) qeyd edib daha sonra hər üç parametrin arqumentlərini topladıq,nəticə ekrana çap olundu

[31]:
```
def adder(x, y, z):
    print("Result:",x + y + z)

adder(4,2,8,5)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-31-66f81e215768> in <module>
      2     print("Result:",x + y + z)
      3
----> 4 adder(4,2,8,5)

TypeError: adder() takes 3 positional arguments but 4 were given
```

Xəta alacağımızı zatən bilirdik.artıq 4-cü elementi ala bilmədi.

[32]:
```
def toplama (*args):
    sums = 0;
    for i in args:
        sums+=i
    print("Result",sums)
toplama(2,4)
```

```
Result 6
```

*args - ifadəsi elementləri kortej daxilində tutduğundan dolayı,for operatoru ilə hər dövrdə i dəyişəninə mənimsədirik

[33]:
```
kortej = (2,3,4,5,7)
for i in kortej:
    print(i)
```

```
2
3
4
5
7
```

kortej olduğuna dair əyani nümunə ilə görmək istəsək

[34]:
```
def printf(*args):
    print(args)
    print(type(args))
printf(1,2,3,4)
```

```
(1, 2, 3, 4)
<class 'tuple'>
```

kimi yaza bilərik

[35]:
```
def toplama (*args):
    sums = 0;
    for i in args:
        sums+=i
    print("Result",sums)
toplama(2,4,5,6,7,8)
```

```
Result 32
```

1.0.5 ** Args ifadəsi

*args ifadəsindən dönən dəyərlər kortej daxilində göstərilirdisə, **args ifadəsindən dönən dəyərlər lüğət data (verilən)tiplərini ifadə edə bilir

[36]:
```
def info(**data):
    print("{}".format(data))
info(Ad="Arzu", Soyad="Nəcəfov", Yaş=40, Mobil='+99455200')
```

```
{'Ad': 'Arzu', 'Soyad': 'Nəcəfov', 'Yaş': 40, 'Mobil': '+99455200'}
```

Yuxarıdakı nümunəyə diqqət edin.qeyd etdiyimiz args ifadəsini data ifadəsi ilə əvəzlədik.fərq etmir siz args əvəzinə istənilən ifadə yaza bilərsiniz.** iki ulduz işarəsi ilə ifadə etdiyimiz arqumentlər dictionary lüğət tipi verilənlərdir.arqumentlər lüğətin açar sözləri,arqument dəyərləri isə həmin açar sözlərin dəyərləridir.

gər lüğət tipində verilən əldə edə biliriksə,deməli lüğət metodlarından istifadə edərək açar söz və dəyərləri rahatlıqla əldə edə bilərik

[37]:
```
def info(**data):
    print("{}".format(data))
info(a="Arzu",b="Nəcəfov")
```

```
{'a': 'Arzu', 'b': 'Nəcəfov'}
```

[38]:
```
def info(**data):
    print("\nData type of argument: ",type(data))

    for key, value in data.items():
        print("{} : {}".format(key, value))

info(Ad="Arzu", Soyad="Nəcəfov", Yaş=40, Mobil='+99455200')
info(Ad="Sahilə", Soyad="Məmmədova", Poçt="example@gmail.com",␣
 ↪Ölkə="Azərbaycan", Yaş=25, Mobil='+9945520')
```

```
Data type of argument:  <class 'dict'>
Ad : Arzu
Soyad : Nəcəfov
Yaş : 40
Mobil : +99455200

Data type of argument:  <class 'dict'>
Ad : Sahilə
Soyad : Məmmədova
Poçt : example@gmail.com
Ölkə : Azərbaycan
Yaş : 25
Mobil : +9945520
```

[ ]:
```

```

[39]:
```
# *args və **args ifadələrini arqument olaraq birbaşa funksiyaya daxil edə␣
 ↪bilərik
```

[40]:
```
def arqument(arg_1, arg_2,arg_3):
    print("arg_1:", arg_1)
    print("arg_2:", arg_2)
    print("arg_3:", arg_3)
args = ("Arqument1", "Arqument2", "Arqument3")
arqument(*args)
```

```
arg_1: Arqument1
arg_2: Arqument2
```

```
arg_3: Arqument3
```

Yuxarıdakı nümunədə 3 parametr qeyd edib args - kortej verilən tipinə 3 element əlavə etdik.Daha sonra funksiyaya *args arqumentini əlavə edərək funksiyanı çalışdırdıq.Nəticədə hər üç element ekrana çap olundu

*args elementləri kortej ilə göstərməsi o demək deyilki biz bunu siyahılara tətbiq edə bilmərik.Nümunələrə baxaq

[41]:
```
def siyahılar(*args):
    my_list = []
    for i in my_list:
        i.append(args)
    print(args)

siyahılar(1,2,3,4,5,6,7,8)
```

```
(1, 2, 3, 4, 5, 6, 7, 8)
```

[48]:
```
def lists(arg1,arg2,arg3):
    print(listlər)
listlər = [*range(3)]
lists(*listlər)
```

```
[0, 1, 2]
```

və ya 3-cü elementi biz daxil edə bilərik

[74]:
```
def lists(arg1,arg2,arg3):
    print(arg1)
    print(arg2)
    print(arg3)
listlər = [2,3]
lists(1, *listlər)
```

```
1
2
3
```

[75]:
```
def funksiya(*args):
    for i in args:
        print(i)
lists = [1, 0.5, 'C++','Python']
funksiya(lists)
```

```
[1, 0.5, 'C++', 'Python']
```

[2]:
```
def topla(arg1,arg2):
    print('Salam'+arg1+arg2)
topla(10,'Elşən')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-568ecda5bd97> in <module>
      1 def topla(arg1,arg2):
      2     print('Salam'+arg1+arg2)
----> 3 topla(10,'Elşən')

<ipython-input-2-568ecda5bd97> in topla(arg1, arg2)
      1 def topla(arg1,arg2):
----> 2     print('Salam'+arg1+arg2)
      3 topla(10,'Elşən')

TypeError: must be str, not int
```

Yuxarıdakı ifadədə xəta aldıq. TypeError: must be str, not int - xətadanda göründüyü kimi verilən tipi xətası.Bilirikki sətir tipi verilən ilə ədəd tipi veriləni toplaya bilmərik

[3]:
```
'python'+10
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-28a23c9b33ad> in <module>
----> 1 'python'+10

TypeError: must be str, not int
```

Yuxarıdakı nümunə funksiyada aldığımız eyni xətadır.

tip çevirmələri və ya type funksiyası ilə verilənin tipini müəyyən edə bilərik

[7]:
```
def topla(arg1):
    if type(arg1) == str:
        print('Salam'+arg1)
    else:
        print('Salam '+str(arg1))
topla(10)
```

```
Salam 10
```

[10]:
```
def topla(arg1):
    if type(arg1) == str:
        print('Salam ' + arg1)
    else:
        print('Salam '+ str(arg1))
topla('Nizami')
```

```
Salam Nizami
```

1.0.6 Funksiyalarda Return ifadəsi

Riyazi əməllərin hesablanmasında nəticə etibarı ilə bir dəyər əldə edirik.Daha sonra bu dəyərdin digər kod bloklarında istifadə etmək üçün məhz funksiyalara üz tuturuq.Amma həmin dəyər funksiya daxilində hesablama əməliyyatından sonra nonetype olaraq qalacaq.Bu nonetype olduğu üçün biz həmin dəyər tipi üzərində digər əməliyyatları apara bilmərik.Bunun üçün dəyəri geri döndürüb nəticəni ədəd tipində əldə edə bilərik.Bunun üçün funksiyalarda return ifadəsi istifadə olunur.

[106]:
```
def square(x):
    y = x * x
    print(y)
```

[109]:
```
S = square(10)
```

```
100
```

[110]:
```
type(S)
```

[110]:
```
NoneType
```

Yuxarıdakı nümunədə yeni olan kod yoxdur,Verilmiş ədədin özünə hasilini əldə etdik.Daha sonra square funksiyasını bir dəyişənə mənimsətdik və tipini soruşduqda noneType ekrana çap olundu.verilənin tipi yoxdur kimi səsləndirək

[113]:
```
def square(x):
    y = x * x
    return y
S = square(10)
print(S)
```

```
100
```

[114]:
```
print(type(S))
```

```
<class 'int'>
```

[ ]:
```

```

Və return ifadəsini istifadə etməklə funksiyadan dönən dəyəri əldə etdik.

[ ]:
```

```

[118]:
```
def minus(n):
    print('Result:',n-50)
minus(S)
```

```
Result: 50
```

Funksiyalar vasitəsilə faktorial hesablanması

[11]:
```
#### Faktorial nədir!
```

1.0.7 1-dən n-ə qədər olan natural ədədlərin hasilinə n- faktorial n-faktorial deyilir. n! kimi işarə olunur.

[13]:
```
print("""

        1! = 1
        2! = 1*2 = 2
        3! = 1*2*3 = 6
        4! = 1*2*3*4 = 24

""")
```

```
        1! = 1
        2! = 1*2 = 2
        3! = 1*2*3 = 6
        4! = 1*2*3*4 = 24
```

Faktorialın hesablanması və iki faktorial ədədin bölünməsi vurulması kimi metodlardan da istifadə etmək olur

[15]:
```
#  3! + 2! = 8
```

[ ]:
```

```

[121]:
```
def fact(n):
    i=1 # başlanğıc dəyərini 0-dan fərqli dəyər.İstənilən ədədin 0-a hasili 0␣
 ↪ədədini verdiyi üçün
    result=1 #
    while i<=n:
        result*=i
        i+=1
    return result
fact(0)
```

[121]: 1

[122]:
```
def fact(n):
    i=1 # başlanğıc dəyərini 0-dan fərqli dəyər.İstənilən ədədin 0-a hasili 0␣
 ↪ədədini verdiyi üçün
```

```
    result=1 #
    while i<=n:
        result*=i
        i+=1
    return result
fact(5)
```

[122]: 120

1.0.8 Recursive Funksiyalar

Funksiya özünü təkrar olaraq çağırırsa recursive funksiyalar adlanır,yəni özünü təkrarlayan funksiyalar. Rekursiv funksiyalar həmçinin dövrlərin əvəzedicisidir

[104]:
```
def topla(say):
    if say == 1:
        return 1
    # Recursive Case
    else:
        return say + topla(say - 1)
```

[105]:
```
topla(7)
```

[105]: 28

Funksiya necə işlədi!

[134]:
```
print("""
        topla(7)
            return 7 + topla(6)
                return 6 + topla(5)
                      return 5 + topla(4)
                            return 4 + topla(3)
                                return 3 + topla(2)
                                    return 2 + topla(1)
                                           return 1


""")
```

```
topla(7)
    return 7 + topla(6)
        return 6 + topla(5)
              return 5 + topla(4)
                    return 4 + topla(3)
                        return 3 + topla(2)
                            return 2 + topla(1)
```

```
return 1
```

Rekursiv funksiyalara baris nümunə ədədin faktorialını funksiyaya daşımaqdır.Yuxarıda yazdığımız faktorial funksiyasını rekursiv olaraq yazaq. Recursive case: n! = n * (n-1)!

[102]:
```
def factorial_recursive(n):
    if n == 1:
        return 1
    else:
        return n * factorial_recursive(n-1)
factorial_recursive(5)
```

[102]: 120

Düstura əsasən n! = n * (n-1)! 5 in faktorialı tapılır, 5-1= n olduğu üçün 4-ün faktorialı tapılır 4-1 =n 3 olduğu üçün üçün faktorialı tapılır 3-1=n 2 olduğu üçün ikinin faktorialı tapılır və 2-1=n 1 olduqda if bloku çalışır və geri dəyər 1 döndürülür blok burada sonlandırılır.

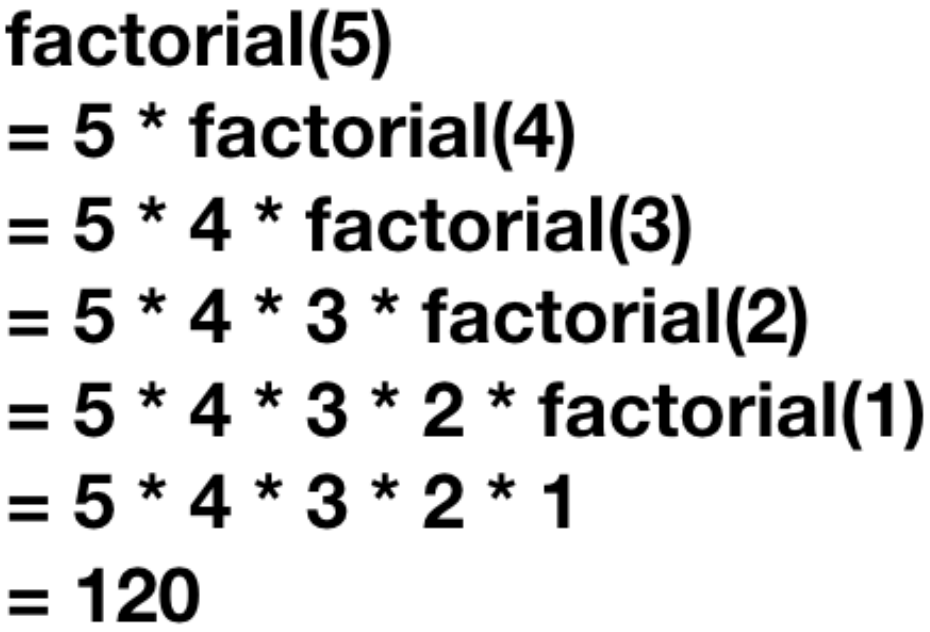
[149]:
```
print("""

4! = 24
3! = 6
2! = 2
1! = 1


""")
```

```
4! = 24
3! = 6
2! = 2
1! = 1
```

5! = 5* 24 = 120

[103]:
```
from IPython.display import Image
Image("image/f.png")
```

[103]:



[4]:
```
def cüt(num):
    #base case
    if num == 0: # limit tətbiq edirik
        return 0
    #recursive case
    else:
        return num + cüt(num - 2)
cüt(10)
```

[4]: 30

Yuxarıdakı nümunədə 10 ədədindən n-2 çıxaraq 0-a doğru azalmanı təşkil edirik,nəticə 0 olduğu vəziyyətdə if şərt operatoru arqument ilə 0-ı müqayisə edir şərt doğrudursa blok sonlanır,10-8-6-4-2-0 = return 10+8+6+4+2+0=30 a bərabərdir

[98]:
```
def negativ(num):
    #base case
    if num == -1: # limit tətbiq edirik
        return -1
    #recursive case
    else:
        return num + negativ(num +1)
negativ(-10)
```

[98]: -55

və Qeyd edimki hər zaman rekursiv funksiyaları istifadə edə bilməzsiniz,rekursiv funksiyalar üçün ədəd hesablamalarında limit tətbiq olunur.Bunu test etmək üçün

[73]:
```
import sys
sys.getrecursionlimit()
# rekursiyanın qiyməti(dərinliyi) əməliyyat sistemindən asılı olaraq dəyişir.Və
 ↪əməliyyat sistemi tərəfindən təyin olunan stek ölçüsünə görə məhdudlaşır
```

[73]:
```
1000
```

sys modulunu import etməklə baxa bilərsiniz.

[101]:
```
def negativ(num):
    #base case
    if num == -1: # limit tətbiq edirik
        return -1
    #recursive case
    else:
        return num + negativ(num +1)
negativ(-1500)
```

```
---------------------------------------------------------------------------
RecursionError                            Traceback (most recent call last)
<ipython-input-101-66a06f90877f> in <module>
      6     else:
      7         return num + negativ(num +1)
----> 8 negativ(-1500)

<ipython-input-101-66a06f90877f> in negativ(num)
      5     #recursive case
      6     else:
----> 7         return num + negativ(num +1)
      8 negativ(-1500)

… last 1 frames repeated, from the frame below …

<ipython-input-101-66a06f90877f> in negativ(num)
      5     #recursive case
      6     else:
----> 7         return num + negativ(num +1)
      8 negativ(-1500)

RecursionError: maximum recursion depth exceeded in comparison
```

“RecursionError: maximum recursion depth exceeded in comparison” formatda xəta alacaqsınız

1.0.9 Global və Lokal Dəyişənlər

Dəyişənlər mövzusunu öyrənərkən dəyişənlər təsnifatına görə " Qlobal "və"lokal" dəyişənlər olaraq iki yerə bölünür.Şərt və dövr operatorları bloklarında operatordan xaricdə və daxildə istifadə etdiyimiz dəyişənlər tək həmin operatora aid deyil.global dəyərlərdir.Bir sözlə dövr və şərt operatorlarında bütün dəyişənlər global dəyişənlərdir.Amma funksiya bloklarında istisnalar mövcuddur. vvəl də qeyd etdikki funksiya bloku daxilində verilənlər,yalnız o funksiyaya mənsub olub,funksiya çağırılmadığı müddətcə verilənlər özlərini ifadə edə bilməz.

[130]:
```
siyahı = []
for i in range(10):

    if i == 2:
        continue;
    siyahı.append(i)
print(siyahı)
```

```
[0, 1, 3, 4, 5, 6, 7, 8, 9]
```

Yuxarıdakı nümunədə siyahı dəyişənini for operatorundan öncə yazsaqda operator bu global dəyişəni görə bilir.Aşağıdakı nümunəyə nəzər yetirin

[136]:
```
def function():
    lists1 = [*range(5)]
    print('Local:',lists1)
    print('-'*30)

function()
print('Global:',lists1)
```

```
Local: [0, 1, 2, 3, 4]
------------------------------
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-136-c976f0a7c060> in <module>
      5
      6 function()
----> 7 print('Global:',lists1)

NameError: name 'lists1' is not defined
```

Yuxarıdakı nümunədə xəta aldıq.NameError: name 'lists1' is not defined xətada deyilirki lists1 qeyd olunmayıb.Bu adda dəyişən mövcud deyil.Bəli funksiyadan kənarda funksiya daxilindəki ifadələr istifadə oluna bilməz

[137]:
```
def funksiya ():
    print(x);
funksiya()
```

```
x= 'Python'
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-137-25696c443e4b> in <module>
      1 def funksiya ():
      2     print(x);
----> 3 funksiya()
      4 x= 'Python'

<ipython-input-137-25696c443e4b> in funksiya()
      1 def funksiya ():
----> 2     print(x);
      3 funksiya()
      4 x= 'Python'

NameError: name 'x' is not defined
```

Yuxarıdakı nümunədə xətanın səbəbini aydınlaşdıraq.Funksiya daxilində x dəyişəninin var olduğunun fərqinə varır,öncə global x dəyərinə baxır və global x təyin olunmayıb,daha sonra daxilində local x dəyərini gözdən keçirir və bu nöqtədə də x-dəyişəninin dəyərini görmür və bunun bir xəta olduğu qənaitinə gəlir.funksiya çağırıldıqdan sonra x dəyişəninə dəyər versək də bunun funksiya üçün heç bir əhəmiyyəti yoxdur

[140]:
```
siyahı = [*range(10)]
def siyahı_ ():
    siyahı = [*range(5)]
    print('Local:',siyahı)
    print('-'*30)
siyahı_()
print('Global:',siyahı)
```

```
Local: [0, 1, 2, 3, 4]
------------------------------
Global: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Yuxarıdakı nümunədə isə funksiyadan kənarda siyahı tərtib etdik.Daha sonra funksiya daxilində yeni siyahı tipində 5-ə qədər ədəd diapazonu təşkil edib print()funksiyası ilə ekrana çap etdik.Funksiyanı çağırırıq və 0-dan 5-ə qədər siyahı çap olunur,funksiyadan sonra yazdığımız print() funksiyası isə siyahı_ funksiyası daxilindəki siyahı elementlərini deyil,global siyahı verilən tipini çap etdi.Funksiya ilk öncə global siyahı olduğunu nəzərdən keçirir daha sonra blokunda başqa siyahı olduğunu görür.Burda qeyd etmək istəyirəmki funksiya global dəyəri deyil öz daxilində əgər qeyd olunmuş eyni adda ifadə varsa onu seçir.Və daxilindək siyahı elementlərini nəzərə alaraq kod blokuna davam edir.Yazdığımız kodlarda iki ayrı dəyişən siyahı verilən tipi mövcuddur.Biri global funksiyadan kənarda,digəri funksiya daxilində local siyahı verilən tipi

[141]:
```
siyahı = [*range(10)]
def siyahı_ ():
    siyahı = [*range(5)]
    siyahı = ['Python']
    print('Local:',siyahı)
    print('-'*30)
siyahı_()
print('Global:',siyahı)
```

```
Local: ['Python']
------------------------------
Global: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

python dilində kodlar sətir sətir oxunur.Eyni adda müxtəlif elementli dəyişən mövcuddursa ən sonuncusun nəzərə alır.Yuxarıdakı nümunədə də ən son siyahı verilən tipi,elementi Python ifadəsi olanı seçir

Funksiya daxilindən global dəyişəni dəyişmək

[143]:
```
kortej = ('Python',)
def kortej_ ():
    global kortej
    kortej = ('C++',)
    print('Local:',kortej)
    print('-'*30)
kortej_()
print('Global:',kortej)
```

```
Local: ('C++',)
------------------------------
Global: ('C++',)
```

kortej tipində elementi Python olan dəyişən təyin etdik.Və bilirikki bu dəyişən global dəyişəndir.Daha sonra funksiya daxilində global kortej dəyişənini çağırıb daha sonra dəyərini C++ ilə əvəzlədik.Bu üsulla funksiyadan xaricdə global kortej dəyişəninin də dəyəri C++ olaraq dəyişildi.Və bu səbəbdən funksiyadan sonra print() funksiyası global kortej dəyişəninin dəyərini c++ olaraq çap edir

Nəticə etibarı ilə təkrar olaraq qeyd edimki şərt və dövr operatorları ilə istifadə olunan dəyişənlər global dəyişənlər,funksiya daxilində istifadə olunan dəyişənlər lokal dəyişənlər olub yalnız həmin funksiyaya məxsusdur

Ümumi olaraq bura qədər öyrəndiyimiz funksiyaların bir neçə arqumentlərini nəzərdən keçirdik. Funksiya arqumentləri 1) Mütləq arqumentlər(Required arguments) 2) Açar sözlü arqumentlər(Keyvord arguments) 3)Susmaya görə arqumentlər(Default arguments) və son olaraq ixtiyari uzunluqlu(limits=256) arqumentlər(Variable-length arguments) olaraq bir neçə metodla yazılır.

[67]:
```
def _Number():
    num1 = float(input('Ilk ədəd:'))
```

```
    num2 = float(input('İkinci ədədi daxil edin:'))
    return num1,num2
def Operation(operation=''):
    print("""

        e(E) Çıxış
        +    Toplama
        -    Çıxma
        *    Vurma
        /    Bölmə
    """)
    operation = input(' məliyyatı seçin:')
    while operation not in ['e','E','+','-','*','/']:
        print(' məliyyatı doğru yazın...!')
    return operation

def _Calculate():
    while True:
        oper1,oper2 = _Number()
        operation = Operation()
        if operation =='+':
            print('Cavab:',oper1+oper2)
        if operation =='-':
            print('Cavab:',oper1-oper2)
        if operation =='*':
            print('Cavab:',oper1*oper2)
        if operation =='/':
            print('Cavab:',oper1/oper2)
        if operation == 'e' or operation == 'E':
            print('Proqramdan çıxılır.....!')
            break
```

[70]:
```
_Calculate()
```

```
Ilk ədəd:12
İkinci ədədi daxil edin:12


        e(E) Çıxış
        +    Toplama
        -    Çıxma
        *    Vurma
```

```
        /    Bölmə

 məliyyatı seçin:*
Cavab: 144.0
Ilk ədəd:4
İkinci ədədi daxil edin:5


        e(E) Çıxış
        +    Toplama
        -    Çıxma
        *    Vurma
        /    Bölmə

 məliyyatı seçin:E
Proqramdan çıxılır…!
```

1.0.10 Anonim Funksiyalar (lambda ifadəsi)

def açar sözündən istifadə edərək funksiyaları tərtib etdik.Python proqramlaşdırma dilində anonim funksiyalar varki lambda açar sözü ilə təyin olunurlar.

[20]:
```
print("""

    lambda arqumentlər: ifadələr



    lambda arguments : expression

""")
```

```
    lambda arqumentlər: ifadələr



    lambda arguments : expression
```

lambda ifadəli funksiyalar əsasən qısa funksiyaları təyin etdiyimiz zaman istifadə edirik.

[2]:
```
sqrt = lambda x,y: x**2+y**2
```

Anonim funksiyanı sqrt - ifadəsinə mənimsətdik.funksiyamız daxilindi iki dəyişən (x,y) qeyd edirik daha sonra iki nöqtə işarəsindən sonra yerinə yetiriləcək əməli yazırıq.iki ədədin qüvvətə yüksəltməsindən alınan nəticənin toplanması funksiyasıdır

[4]:
```
sqrt(2,3) # 4 + 9 = 13
```

[4]:
```
13
```

və ya print()-funksiyası ilə birbaşa ekrana çap edə bilərik

[9]:
```
print((lambda i: i*0.5) (10))
```

```
5.0
```

1.0.11 Nümunələr

[12]:
```
var = lambda x, y: x + y
print ('Result:',var (1, 2))
```

```
Result: 3
```

[13]:
```
string='Bakı Azərbaycanın paytaxtıdır.'
print(lambda string : print(string))
```

```
<function <lambda> at 0x7ffa2438d830>
```

[18]:
```
string='Bakı Azərbaycanın paytaxtıdır.'
(lambda string : print(string))(string)
```

```
Bakı Azərbaycanın paytaxtıdır.
```

[22]:
```
x = lambda a : a + 10
print(x(2))
```

```
12
```

[24]:
```
x = lambda a, b, c : a + b + c # a,b,c adlı 3 dəyişən qeyd edib hər üç␣
↪dəyişənin dəyərini toplayırıq
print(x(1,2,3))
```

```
6
```

[35]:
```
[(lambda x: x )(x) for x in range(10)]
```

[35]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[38]:
```
squares = []
for x in range(10):
    print(x)
    squares.append(x*x)
print(squares)
```

```
0
1
2
3
4
5
6
7
8
9
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Yuxarıdakı nümunədə 0-dan 10-a qədər ədədlər diapazonu tərtib edib siyahı və hər dövrdə ədədin özünə hazilindən alınan qiyməti squares-siyahısına əlavə edirik.

və bir də yuxarıdakı nümunənin lambda ilə yazılışına baxaq

[40]:
```
[(lambda x: x*x)(x) for x in range(10)]
```

[40]:
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

1.0.12 Funksiyaların sənədləşdirilməsi

Funksiyalar bəhsinə keçid edərkən,funksiyalar daxilində docstring istifadəsindən danışmışdıq.Bu ifadə funksiya haqqında məlumat toplusu olub funksiyanın hansı işləri görəcəyinə dair məlumat verir.Bu ifadəni yazmaqla kod blokunun daxilində ilk öncə qarşımıza çıxdığından,tərtib etdiyimiz funksiyanı həm biz həm də başqaları çalışdırmadan öncə məlumatlanır.

Həmçinin ifadə (” **doc** “) adı ilə funksiya daxilində obyekt ilə təmsil olunur.

[77]:
```
def factorial(n):
    """
    Returns factorial of a given number using recursion.

    fact(1) -------> 1

    """
    if n <= 1:
        return 1
    else:
        return (n*factorial(n-1))
```

[78]:
```
factorial(5)
```

[78]:
```
120
```

[79]:
```
print(factorial.__doc__)
```

```
    Returns factorial of a given number using recursion.
```

```
fact(1) -------> 1
```

[ ]: