توابع

مقدمه ای بر توابع

این سخنرانی شامل توضیح این است که یک تابع در پایتون چیست و چگونه می توان آن را ایجاد کرد. توابع یکی از اصلی ترین بلوک های ساختمانی ما خواهند بود زمانی که ما مقدار بیشتر و بیشتری از کد را برای حل مشکلات بسازیم.

یک تابع چیست؟

به طور رسمی، یک تابع یک دستگاه مفید است که مجموعه ای از عبارات را در کنار هم قرار می دهد تا بتوانند بیش از یک بار اجرا شوند. آنها همچنین به ما اجازه می دهند که پارامترهایی را مشخص کنیم که به عنوان ورودی های توابع عمل کنند.

در سطح بنیادی تر، توابع به ما اجازه می دهند که نیاز نباشد همان کد را دوباره و دوباره تکرار کنیم. اگر به درس های قبل درباره رشته ها و لیست ها برگردید، به خاطر داشته باشید که ما از یک تابع `len()` برای دریافت طول یک رشته استفاده کردیم. از آنجایی که بررسی طول یک دنباله یک وظیفه رایج است، شما مایلید یک تابع بنویسید که این کار را در هر فرمان تکرار کند.

توابع یکی از سطوح پایه ای استفاده مجدد از کد در پایتون خواهند بود و همچنین به ما اجازه می دهد تا شروع به فکر کردن در مورد طراحی برنامه کنیم (ما بسیار عمیق تر در مورد ایده های طراحی وقتی درباره برنامه نویسی شی گرا یاد می گیریم، غوطه ور خواهیم شد).

چرا حتی از توابع استفاده کنید؟

به طور ساده، شما باید از توابع استفاده کنید وقتی قصد دارید چندین بار از یک بلوک کد استفاده کنید. تابع به شما اجازه می دهد تا همان بلوک کد را فراخوانده شود بدون اینکه نیاز باشد آن را چندین بار بنویسید. این نوبت به شما اجازه می دهد تا اسکریپت های پایتون پیچیده تری ایجاد کنید. برای درک واقعی این، ما واقعاً باید توابع خودمان را بنویسیم!

مباحث تابع

- کلمه کلید def
- نمونه ساده از یک تابع
- فراخوانی یک تابع با ()
- قبول پارامتر
- فراخوانی بصورت keyword argument
- مقدار پیش فرض برای پارامتر ورودی
- ارسال مقدار vs ارسال رفرنس
- چاپ در مقابل بازگشت
- بازگشت همزمان چند مقدار
- اضافه کردن منطق در داخل یک تابع
- اضافه کردن حلقه ها در داخل یک تابع
- توابع با تعداد ووردی دلخواه *args
- توابع با تعداد و نام ورودی دلخواه **kwargs
- تعاملات بین توابع

In [ ]:

```
s = 'foobar'

len(s)
```

In [ ]:

```
s.upper()
```

In [ ]:

```
id(s)
```

کلمه کلیدی def

بیایید ببینیم چگونه ساختار یک تابع در پایتون را ایجاد کنیم. این شکل زیر را دارد:

In [ ]:

```
def name_of_function(arg1,arg2):
    '''
    This is where the function's Document String (docstring) goes.
    When you call help() on your function it will be printed out.
    '''
    # Do stuff here
    # Return desired result
```

ما با `def` شروع می کنیم سپس یک فضای خالی پس از نام تابع. سعی کنید نام ها را مرتبط نگه دارید، به عنوان مثال `len()` یک نام خوب برای یک تابع `length()` است. همچنین با نام ها مراقب باشید، شما نمی خواهید یک تابع را با همان نام یک [تابع داخلی در پایتون](https://docs.python.org/3/library/functions.html) (مانند len) نامگذاری کنید.

سپس یک جفت پرانتز با تعدادی آرگومان جدا شده توسط یک کاما می آید. این آرگومان ها ورودی های تابع شما هستند. شما قادر خواهید بود از این ورودی ها در تابع خود استفاده کنید و به آنها ارجاع دهید. پس از این شما یک کولن قرار می دهید.

حالا مرحله مهم، شما باید برای شروع به درستی کد درون تابع خود را indent کنید. پایتون از *فضای سفید* برای سازماندهی کد استفاده می کند. بسیاری از زبان های برنامه نویسی دیگر این کار را انجام نمی دهند، بنابراین به آن فکر کنید.

سپس شما docstring را مشاهده خواهید کرد، این جایی است که شما یک توضیحات ابتدایی از تابع را می نویسید. با استفاده از Jupyter و Jupyter Notebooks، شما قادر خواهید بود با فشار دادن Shift+Tab پس از نام تابع این docstrings را بخوانید. Docstrings برای توابع ساده ضروری نیستند، اما تمرین خوبی است که آنها را درون قرار دهید تا شما یا دیگران به راحتی کدی را که نوشته اید درک کنید.

پس از همه اینها شما شروع به نوشتن کدی می کنید که مایلید اجرا شود.

بهترین راه برای یادگیری توابع، عبور از مثال هاست. پس بیایید سعی کنیم از مثال هایی عبور کنیم که به اشیاء و ساختارهای داده ای که قبلاً یاد گرفته ایم مرتبط باشد.

نمونه ساده ای از یک تابع

In [3]:

```
def f():
    s = '__ Inside of f()'
    print(s)
```

فراخوانی یک تابع call

In [4]:

```
f()
```

```
__ Inside of f()
```

اگر شما پرانتز () را فراموش کنید، آن را به سادگی نشان می دهد که say_hello یک تابع است. بعداً ما یاد خواهیم گرفت که در واقع می توانیم توابع را در داخل توابع دیگر قرار دهیم! اما در حال حاضر، فقط به خاطر داشته باشید که توابع را با () فراخوانی کنید.

In [5]:

```
f()
```

```
__ Inside of f()
```

In [6]:

```
f
```

Out[6]:

```
<function __main__.f()>
```

In [7]:

```
for i in range(10):
    f()
```

```
__ Inside of f()
__ Inside of f()
__ Inside of f()
__ Inside of f()
__ Inside of f()
__ Inside of f()
__ Inside of f()
__ Inside of f()
__ Inside of f()
__ Inside of f()
```

1.2) ارسال ارگومان ورودی

In [8]:

```
def add(a, b, c):
    result = a + b + c
    print(f'Result: {result}')
```

In [9]:

```
add(3, 7, 18)
```

```
Result: 28
```

نکته مهم:

به متغیرهایی همچون a,b,c که در امضا تابع مشخص می شوند پارامتر ورودی می گویم، و به مقادیر 3,7,8 که در زمان فراخوانی تابع بعنوان ورودی به آن ارسال می شود آرگومان ورودی می گوییم.

In [10]:

```
add(5, 24, 109)
```

```
Result: 138
```

In [13]:

```
def add1(a: int, b: int, c: int) -> int:
    result = a + b + c
    print(f"Result: {result}")
```

In [14]:

```
add1(5, 24, 109)
```

```
Result: 138
```

In [15]:

```
add1('a', 'b', 'c')
```

```
Result: abc
```

In [16]:

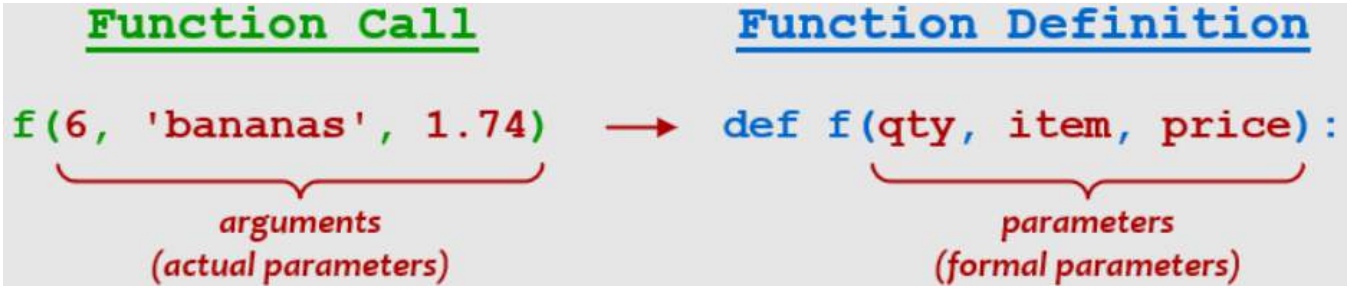
```
def f(qty, item, price):
    per_item = price / qty
    print(f'{qty} of {item} cost ${price: .2f}')
```

در فراخوانی زیر ما فقط مقادیر را به تابع ارسال می کنیم و هیچ گونه اطلاعات دیگری را به در مورد فراخوانی به پایتون نمی دهیم، از این رو، پایتون مقادیر را بصورت موقعیت مکانی به ترتیب از سمت چپ داخل پارامترها قرار می دهد.

In [17]:

```
f(6, 'bananas', 1.74)
```

```
6 of bananas cost $ 1.74
```



In [18]:

```
1 f('bananas', 6, 1.74)
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
Cell In[18], line 1
----> 1 f('bananas', 6, 1.74)

Cell In[16], line 2, in f(qty, item, price)
      1 def f(qty, item, price):
----> 2     per_item = price / qty
      3     print(f'{qty} of {item} cost ${price: .2f}')

TypeError: unsupported operand type(s) for /: 'float' and 'str'
```

In [19]:

```
1 f(6, 1.74)
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
Cell In[19], line 1
----> 1 f(6, 1.74)

TypeError: f() missing 1 required positional argument: 'price'
```

In [20]:

```
1 f('bananas', 6, 1.74, 55)
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
Cell In[20], line 1
----> 1 f('bananas', 6, 1.74, 55)

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

1.2.2) ارسال ارگومان های ورودی بصورت نام دار یا keyword

In [21]:

```
1 f(qty = 11, item = 'Orange', price = 17)
```

```
11 of Orange cost $ 17.00
```

In [22]:

```
f(item = 'Orange', qty = 11, price = 17)
```

```
11 of Orange cost $ 17.00
```

In [23]:

```
f(item = 'Orange', qty = 11, cost = 17)
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
Cell In[23], line 1
----> 1 f(item = 'Orange', qty = 11, cost = 17)

TypeError: f() got an unexpected keyword argument 'cost'
```

1.2.3) مقادیر پیش فرض

گاهی برای انعطاف پذیری بیشتر یک تابع در فراخوانی، امکانی را فراهم می کنیم تا اگر مقدار یک پارامتر در زمان فراخوانی مشخص نشود، آن پارامتر از یک مقدار پیش فرض استفاده نماید.

In [24]:

```
def f(qty, item, price = 5):
    per_item = price / qty
    print(f'{qty} of {item} cost ${price: .2f}')
```

In [25]:

```
f(qty = 6, item = 'bananas', price = 1.74)
```

```
6 of bananas cost $ 1.74
```

In [26]:

```
f(qty = 6, item = 'bananas')
```

```
6 of bananas cost $ 5.00
```

In [ ]:

```
f(6, 'orange')
```

In [27]:

```
def f(qty = 6, item = 'Oranges', price = 2):
    print(f'{qty} {item} cost ${price:.2f}')
```

In [28]:

```
1 f()
```

```
6 Oranges cost $2.00
```

In [29]:

```
1 f(10)
```

```
10 Oranges cost $2.00
```

In [30]:

```
1 f(10, 'bananas')
```

```
10 bananas cost $2.00
```

In [31]:

```
1 f(10, 'bananas', 15)
```

```
10 bananas cost $15.00
```

In [32]:

```
1 f(price = 20)
```

```
6 Oranges cost $20.00
```

In [33]:

```
1 f(price = 20, qty = 5)
```

```
5 Oranges cost $20.00
```

نکته مهم: زمانی که یک مقدار از نوع داده قابل جهش یعنی لیست، دیکشینری و یا مجموعه بعنوان مقدار پیش فرض استفاده می نمایید، هر تغییری که در دفعات پیشین استفاده از مقدار پیش فرض اعمال شده باشد، اثر خود را بر مقدار پیش فرض حفظ خواهد کرد.

In [36]:

```
1 def f(my_list = []):
2     my_list.append('###')
3     return my_list
```

In [37]:

```
1 f(['foo', 'bar', 'baz'])
```

Out[37]:

```
['foo', 'bar', 'baz', '###']
```

In [38]:

```
1 f([1,2,3,4,5])
```

Out[38]:

```
[1, 2, 3, 4, 5, '###']
```

In [39]:

```
1 f()
```

Out[39]:

```
['###']
```

In [40]:

```
1 f()
```

Out[40]:

```
['###', '###']
```

In [41]:

```
1 f()
```

Out[41]:

```
['###', '###', '###']
```

In [42]:

```
1 f([1,2,3,4,5])
```

Out[42]:

```
[1, 2, 3, 4, 5, '###']
```

In [43]:

```
1 f()
```

Out[43]:

```
['###', '###', '###', '###']
```

1.3) Pass-By-Value vs Pass-By-Reference in Python

1.3) ارسال ارگومان ها به توابع بصورت مقدار در مقابل ارسال با ارجاع

آیا پارامترها در پایتون با ارجاع یا با مقدار ارسال می شوند؟ جواب این است که هیچ کدام دقیقاً نیستند. زیرا ارجاع در پایتون به معنای همان چیز در پاسکال نیست.

In [44]:

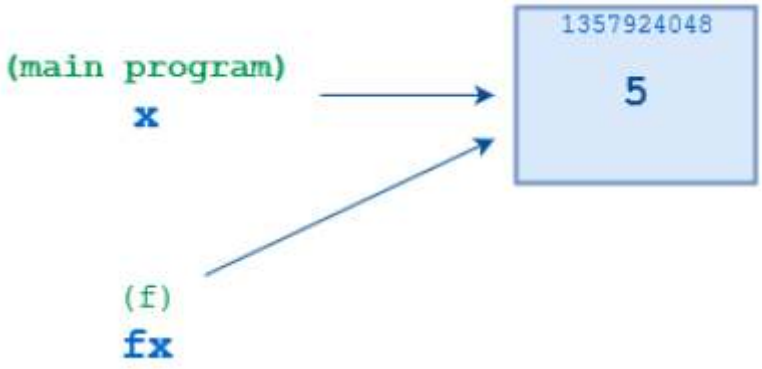
```
def f(fx):
    fx = 10
```

In [45]:

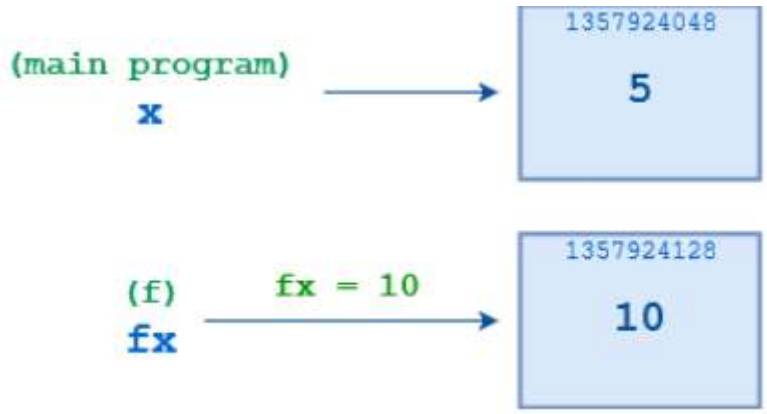
```
x = 5

f(x)

print(x)
```

```
5
```

1) در لحظه فراخوانی تابع "f"



2) بعد از فراخوانی تابع "f"



In [47]:

```
def f(fx):
    print('fx = ', fx, '/ id(fx) = ', id(fx))
    fx = 10
    print('fx = ', fx, '/ id(fx) = ', id(fx))

x = 5
print('1) ===>   x =', x, '/ id(x) = ', id(x))

f(x)

print('2) ===>   x =', x, '/ id(x) = ', id(x))
```

```
1) ===>   x = 5 / id(x) =  2078305960752
fx =  5 / id(fx) =  2078305960752
fx =  10 / id(fx) =  2078311418608
2) ===>   x = 5 / id(x) =  2078305960752
```

In [48]:

```
def my_list(l1):
    print('l1 =', l1, '/ id(l1) = ', id(l1))
    l1 = [11, 12, 13, 14]
    print('l1 =', l1, '/ id(l1) = ', id(l1))


l0 = [5, 6, 7, 8]
print('1) ===>   l0 =',l0, '/ id(l0) = ', id(l0))

my_list(l0)


print('2) ===>   l0 =', l0, '/ id(l0) = ', id(l0))
```

```
1) ===>   l0 = [5, 6, 7, 8] / id(l0) =  2078413703360
l1 = [5, 6, 7, 8] / id(l1) =  2078413703360
l1 = [11, 12, 13, 14] / id(l1) =  2078413712704
2) ===>   l0 = [5, 6, 7, 8] / id(l0) =  2078413703360
```

In [49]:

```
def my_list(l1):
    print('l1 =', l1, '/ id(l1) = ', id(l1))
    l1[0] = 11
    print('l1 =', l1, '/ id(l1) = ', id(l1))


l0 = [5, 6, 7, 8]
print('1) ===>   l0 =',l0, '/ id(l0) = ', id(l0))

my_list(l0)


print('2) ===>   l0 =', l0, '/ id(l0) = ', id(l0))
```

```
1) ===>   l0 = [5, 6, 7, 8] / id(l0) =  2078413388864
l1 = [5, 6, 7, 8] / id(l1) =  2078413388864
l1 = [11, 6, 7, 8] / id(l1) =  2078413388864
2) ===>   l0 = [11, 6, 7, 8] / id(l0) =  2078413388864
```

In [50]:

```
l = [1,2,3]
l1 = l

l1[0] = -100

print(l, l1)
```

```
[-100, 2, 3] [-100, 2, 3]
```

In [51]:

```
def my_list(l1):
    print('l1 =', l1, '/ id(l1) = ', id(l1))
    l1[0] = 11
    print('l1 =', l1, '/ id(l1) = ', id(l1))


l0 = [5, 6, 7, 8]
print('1) ===>   l0 =',l0, '/ id(l0) = ', id(l0))

my_list(l0.copy())


print('2) ===>   l0 =', l0, '/ id(l0) = ', id(l0))
```

```
1) ===>   l0 = [5, 6, 7, 8] / id(l0) =  2078413718720
l1 = [5, 6, 7, 8] / id(l1) =  2078384502720
l1 = [11, 6, 7, 8] / id(l1) =  2078384502720
2) ===>   l0 = [5, 6, 7, 8] / id(l0) =  2078413718720
```

1.4) دستور return برای بازگرداندن مقادیر از تابع

In [54]:

```
def f():
    print('foo')
    print('bar')
    return

type(f())
```

```
foo
bar
```

Out[54]:

```
NoneType
```

In [58]:

```
def f(x):
    if x < 0:
        return
    if x > 100:
        return
    print(x)


f(55)
```

```
55
```

In [60]:

```
def f():
    return 'foo'

s = f()

print(s)

type(f())
```

```
foo
```

Out[60]:

```
str
```

In [63]:

```
def add(a, b, c):
    result  = a + b + c
    return result

r = add(10, 5, 9)

print(r ** 3)
```

13824

In [ ]:

```
def add(a, b, c):
    summation  = a + b + c
    avg = summation / 3
    return summation

r = add(10, 5, 9)

print(r ** 3)
```

1.4.1) بازگرداندن چند مقدار بصورت همزمان از تابع

توجه نمایید که دستور return همواره یک مقدار یا یک بسته را می تواند بعنوان خروجی تابع برگرداند، از این رو، برای بازگرداند چندین مقدار بصورت همزمان در پایتون باید مقادیر مد نظر را در قالب یک بسته بندی مانند لیست، دیکشینری و یا تاپل انجام دهیم.

In [72]:

```
# dictionary

def f():
    return dict(foo = 1, bar = 2, baz = 3)

f()

type(f())

#d = f()

#d['baz']**3

f()['baz'] ** 3
```

Out[72]:

27

In [73]:

```
# return list

def f():
    return ['foo', 'bar', 'baz', 'qux']

f()
```

Out[73]:

```
['foo', 'bar', 'baz', 'qux']
```

In [74]:

```
f()[2]
```

Out[74]:

```
'baz'
```

In [75]:

```
f()[::-1]
```

Out[75]:

```
['qux', 'baz', 'bar', 'foo']
```

In [ ]:

```
x = 1, 2, 3, 4
```

In [76]:

```
# return tuple with packing

def f():
    return 'foo', 'bar', 'baz', 'qux'

f()

```

Out[76]:

```
('foo', 'bar', 'baz', 'qux')
```

In [77]:

```
a, b, c, d = f()

print(f'a = {a}, b = {b}, c = {c}, d = {d}')
```

```
a = foo, b = bar, c = baz, d = qux
```

1.5) توابع با تعداد آرگومان ورودی دلخواه *args

In [80]:

```
def avg(a, b, c):
    return (a + b + c) / 3
```

In [81]:

```
avg(1,2,3)
```

Out[81]:

```
2.0
```

In [83]:

```
avg(1, 5, 10, 4, 6)
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
Cell In[83], line 1
----> 1 avg(1, 5, 10, 4, 6)

TypeError: avg() takes 3 positional arguments but 5 were given
```

In [ ]:

```
def avg(a, b=0, c=0, d=0, e=0):
    .
    .
    .
```

In [ ]:

```
avg(1)
avg(1, 2)
avg(1, 2, 3)
avg(1, 2, 3, 4)
avg(1, 2, 3, 4, 5)
```

In [ ]:

```
def avg(a, b=0, c=0, d=0, e=0):
    num = 1
    if b != 0:
        num += 1

    return (a + b + c + d + e) / # Divided by what???
```

راه حل اول:

In [84]:

```
def avg(a):

    total = 0

    for v in a:
        total = total + v

    return total / len(a)
```

In [86]:

```
avg([1,2,3])
```

Out[86]:

2.0

In [87]:

```
avg([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Out[87]:

5.0

In [88]:

```
t = (1, 2, 3, 4, 5)
avg(t)
```

Out[88]:

3.0

1.5.1) استفاده از بسته بندی تاپل ها برای ساخت توابع با تعداد آرگومان ورودی دلخواه

In [89]:

```
def f(*args):
    print(args)
    print(type(args), len(args))
    for x in args:
        print(x)
```

In [90]:

```
f(1, 2, 3, 4, 5)
```

```
(1, 2, 3, 4, 5)
<class 'tuple'> 5
1
2
3
4
5
```

In [91]:

```
f('foo', 'bar', 'baz', 'qux', 'quux')
```

```
('foo', 'bar', 'baz', 'qux', 'quux')
<class 'tuple'> 5
foo
bar
baz
qux
quux
```

In [92]:

```
def avg(*args):
    total = 0
    for i in args:
        total = total + i
    return total / len(args)
```

In [93]:

```
avg(1, 2, 3)
```

Out[93]:

```
2.0
```

In [94]:

```
avg(1, 2, 3, 4, 5)
```

Out[94]:

```
3.0
```

In [103]:

```
def avg(*args):
    return sum(args) / len(args)

avg(1, 2, 3, 4, 5)
```

Out[103]:

```
3.0
```

1.5.2) استفاده از بازگشایی تاپل ها، لیست ها در فراخوانی توابع

In [96]:

```
def f(x, y, z):
    print(f'x = {x}')
    print(f'y = {y}')
    print(f'z = {z}')
```

In [97]:

```
f(1,2,3)
```

```
x = 1
y = 2
z = 3
```

In [99]:

```
t = ('foo', 'bar', 'baz')

f(*t)
```

```
x = foo
y = bar
z = baz
```

In [100]:

```
a = ['foo', 'bar', 'baz']
f(*a)
```

```
x = foo
y = bar
z = baz
```

In [101]:

```
s = {1, 2, 3}
print(type(s))

f(*s)
```

```
<class 'set'>
x = 1
y = 2
z = 3
```

1.6) ساخت توابع با تعداد و نام ارگومان دلخوه به روش بسته بندی دیکیشنری **kwargs

In [104]:

```
def f(**kwargs):
    print(kwargs)
    print(type(kwargs), len(kwargs))

    for key, val in kwargs.items():
        print(key,  ' -> ', val)

    print(kwargs.get('foo'))
```

In [107]:

```
f(zoo = 1, bar = 2, baz = 3)
```

```
{'zoo': 1, 'bar': 2, 'baz': 3}
<class 'dict'> 3
zoo  ->  1
bar  ->  2
baz  ->  3
None
```

In [108]:

```
f(foo=1, bar=2, baz=3, qux = 351, quux = -197)
```

```
{'foo': 1, 'bar': 2, 'baz': 3, 'qux': 351, 'quux': -197}
<class 'dict'> 5
foo  ->  1
bar  ->  2
baz  ->  3
qux  ->  351
quux  ->  -197
1
```

1.6.1) استفاده از بازگشایی دیکشینری در فراخوانی توابع

In [109]:

```
def f(a, b, c):
    print(f'a = {a}')
    print(f'b = {b}')
    print(f'c = {c}')
```

In [110]:

```
d = {'a': 'foo', 'b': 25, 'c': 'qux'}

f(**d)
```

```
a = foo
b = 25
c = qux
```

In [111]:

```
f(a='foo', b=25, c='qux')
```

```
a = foo
b = 25
c = qux
```

1.7) مثالی از ترکیب همه حالات تعریف تابع

In [112]:

```
def f(a, b, *args, **kwargs):
    print(F'a = {a}')
    print(F'b = {b}')
    print(F'args = {args}')
    print(F'kwargs = {kwargs}')
```

In [113]:

```
f(1, 2, 'foo', 'baz', dd = 'qux', x=100, y=200, z=300)
```

```
a = 1
b = 2
args = ('foo', 'baz')
kwargs = {'dd': 'qux', 'x': 100, 'y': 200, 'z': 300}
```

In [114]:

```
f(1, 2, 'foo', ff = 'bar', 'baz', dd = 'qux', x=100, y=200, z=300)
```

```
  Cell In[114], line 1
    f(1, 2, 'foo', ff = 'bar', 'baz', dd = 'qux', x=100, y=200, z=300)
                                                                      ^
SyntaxError: positional argument follows keyword argument
```

مثال 1

تابع بنویسید تا بررسی کند که لیست ورودی حداقل دارای یک عدد زوج می باشد یا نه

In [119]:

```
def check_even_list(num_list):
    # Go through each number
    for number in num_list:
        # Once we get a "hit" on an even number, we return True
        if number % 2 == 0:
            return True
        # Don't do anything if its not even
        else:
            pass
    # Notice the indentation! This ensures we run through the entire for loop
    return False
```

In [124]:

```
check_even_list([1,3,5,7,9])
```

Out[124]:

```
False
```

تکمیلی: یک تابع بنویسید که لیست تمام اعداد زوج را از داخل لیست ورودی استخراج کند!

In [ ]:

```

```

مثال 2

یک تابع بنویسید تا اول بودن یک عدد را بررسی نماید.

In [125]:

```
def is_prime(num):
    '''
    Naive Method of checking for primes
    '''
    for n in range(2, num):
        if num % n == 0:
            print(f'{num} is not a prime!')
            break
    else:
        print(f'{num} is a Prime Number!!')

```

In [136]:

```
is_prime(5915587277)
```

```
5915587277 is a Prime Number!!
```

In [137]:

```python
import math

def is_prime(num):
    '''
    Better method of checking for primes.
    '''

    if num % 2 == 0 and num > 2:
        return False
    for i in range(3, int(math.sqrt(num)) + 1, 2):
        if num % i == 0:
            return False
    return True

```

In [138]:

```python
is_prime(5915587277)
```

Out[138]:

```
True
```