مقدمه:

لزوما حجم کد بیشتر به معنای برنامه‌ی بهتر نیست! یک برنامه‌نویس خوب یک برنامه بزرگ را در ابتدا به تعدادی زیر مساله تبدیل کرده و سپس به حل هر زیر مساله به صورت مجزا می‌پردازد. به این ترتیب احتمالا این قسمت های کوچک بعدا در همین برنامه یا برنامه های دیگر استفاده خواهد شد و علاوه بر کاهش حجم کد، برنامه نویس با یک بار تست و آزمودن هر قسمت از کد می‌تواند از کارایی آن کد اطمینان حاصل کرده و به بررسی سایر قسمت های برنامه اش بپردازد.

یکی از راهکارهای برنامه نویسی برای محقق شدن این امر توابع هستند. وقتی خطوط برنامه ما زیاد می شود درک، پیگیری، خطایابی و دیگر اعمال بر روی برنامه دشوار خواهد شد. توابع ابزاری هستند که به ما در بهبود برنامه کمک می کنند و برنامه نویسی ساخت یافته را ارائه می دهند، بدین معنا که برنامه اصلی به قسمتهای منطقی و مستقل کوچکتری تقسیم می شود که توابع نام دارند .

هدف این است که بعد از یک بار نوشتن کد یک تابع و تست و اطمینان از صحت آن، به آن تابع به صورت یک جعبه سیاه نگاه کنیم. برای مثال یک ویدیو پرژکتور را بدون آنکه اطلاعی از جزئیات مدارات داخلی آن داشته باشید تنها با اطلاعات پایه ای از ورودی و خروجی ها استفاده می‌کنید. شما تنها از این ویدیو پرکتوری که قبلا ایجاد شده استفاده میکنید و اصلا نیازی ندارید به داخل آن دسترسی پیدا کنید. این ایده در دنیای برنامه نویسی با استفاده از توابع پیاده سازی می‌شود. شما پس از نوشتن یک تابع نیازی ندارید از کدهای داخل آن تابع اطلاعی داشته باشید. تنها و تنها عملکرد، ورودی ها و خروجی آن تابع برایتان مهم خواهد بود.

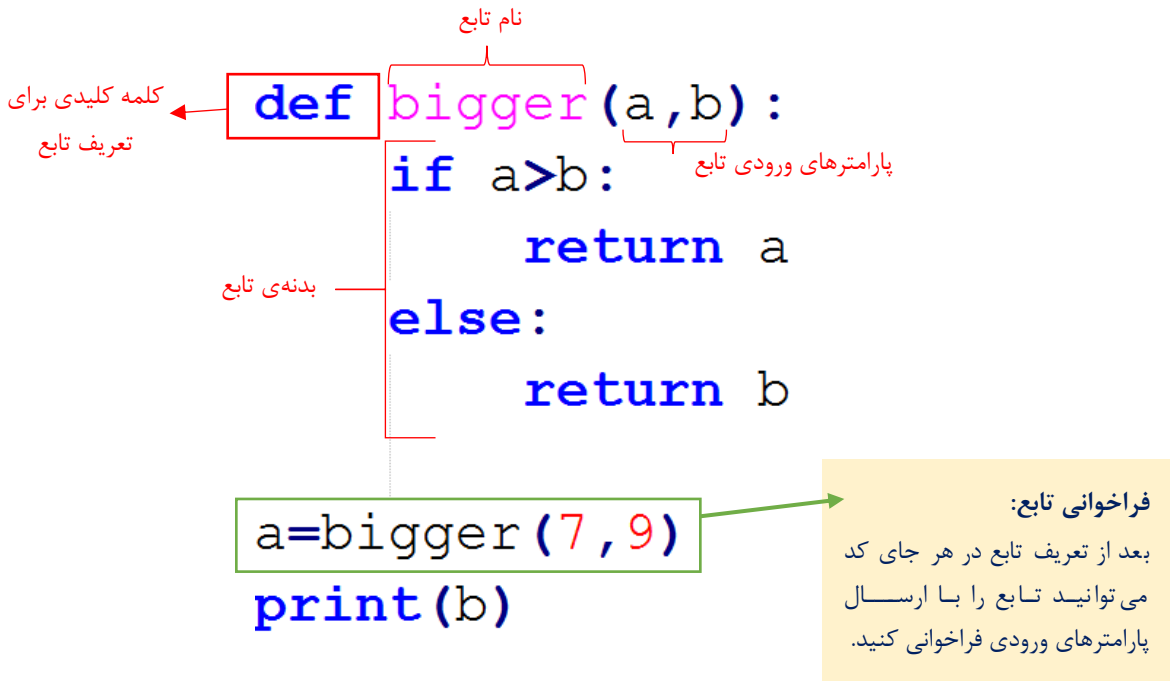
توابع:

- توابعی که در پایتون می‌نویسید تا زمانی که فراخوانی نشوند اجرا نمی‌شوند.
- هر تابع دارای خصوصیات زیر است:
  - نام دارد
  - پارامتر های ورودی دارد (۰، ۱ یا بیشتر)
  - بدنه دارد.

مثال) تابعی بنویسید که سه ورودی بگیرد؛ ورودی اول نام، ورودی دوم نام خانودگی و ورودی سوم یک مقدار منطقی. اگر ورودی سوم True بود، ابتدا نام خانوادگی و سپس نام، در غیر این صورت نام و نام خانوادگی را روی صفحه چاپ کند:

```
def printName(firstName, lastName, reverse):
    if reverse:
        print(lastName + ', ' + firstName)
    else:
        print(firstName, lastName)
```

حال به مثال زیر که عدد بزرگتر را برمی‌گرداند دقت کنید. اجزای یک تابع تشریح شده است:



اگر درون بدنه‌ی تابع را در نظر بگیریم، از عبارت یا عباراتی برای محاسبه یا انجام عملیات درخور برای آن تابع تشکیل شده است.

کلمه‌ی کلیدی return یک مقدار را به عنوان خروجی تابع برمی‌گرداند.

مثال) برنامه‌ای بنویسید که دو عدد به عنوان ورودی گرفته و عدد بزرگتر را برگرداند:

```
def bigger(a,b):
    if a>b:
        return a
    else:
        return b
```

مثال) برنامه‌ی قبل را با عدد ۷ و ۹ فراخوانی کنید و پاسخ را چاپ کنید:

```
a=bigger(7,9)
print(a)
```

این برنامه عدد ۹ را چاپ خواهد کرد.

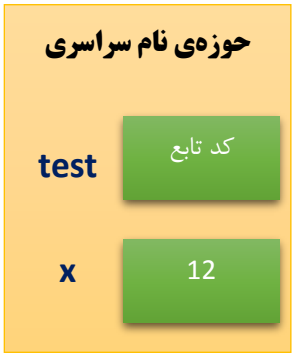
مثال) برنامه‌ای بنویسید که با ۲ بار فراخوانی تابع bigger‌ی که قبلا نوشته شده؛ بزرگترین عدد از بین ۳ عدد ۵ و ۱۱ و ۳ چاپ کند.

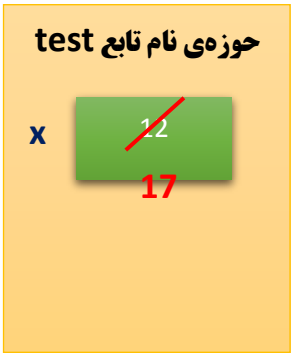
```
a=bigger(5,11)
b=bigger(a,3)
print(b)
```

حوزه های نام در فراخوانی تابع

هنگامی که یک تابع را با پارامتر یا پارامترهایی فراخوانی می‌کنید، مستقل از نامی که هنگام فراخوانی تابع به عنوان پارامتر به آن تابع ارسال شده، مقدار آن متغیر برای آن تابع کپی شده و تغییراتی که آن تابع روی متغیر می‌دهد روی متغیر اصلی اعمال نمی‌شود. به مثال زیر دقت کنید:

```
def test(x):
    x+=5
x=12
print(x)
test(x)
print(x)
```



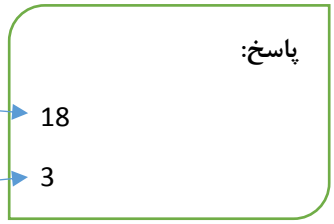


همان‌طور که در تصویر بالا مشاهده می‌شوید، حوزه ی نام تابع test از حوزه‌ی نام سراسری جدا است و تغییراتی که در تابع test روی پارامتری دریافتی x اعمال می‌کند، روی متغیر اصلی اثری ندارد. در مثال بالا قبل از فراخوانی تابع x با مقدار ۱۲ فراخوانی شده، در نتیجه تابع print اول مقدار ۱۲ را چاپ می‌کند. در خط بعدی تابع test با پارامتر x فراخوانی شده است. اما تغییراتی که در آن تابع روی این پارامتر اعمال می‌شود، اثری روی x اصلی خارج از تابع نمی‌گذارد، در نتیجه print دوم در خط آخر نیز همان مقدار ۱۲ را چاپ می‌کند.

توجه کنید که اگر در تابع مقدارx را چاپ می‌کردیم تغییرات مشاهده می‌شد.

مثال) پس از اجرای برنامه ی زیر چه چیزی روی صفحه چاپ می‌گردد؟

```
def test(x):
    x+=15
    print(x)
x=3
test(x)
print(x)
```



هیچ دلیلی ندارد پارامتری که به تابع ارسال می‌شود، هم نام پارامتر ورودی تابع باشد، در مثال قبل پارامتر ورودی تابع x بوده، و هنگام فراخوانی هم متغیر x به تابع ارسال شده بود، اما می‌توان برنامه‌ی قبل را به صورت زیر نوشت و هیچ تقاوتی نخواهد کرد:

```
def test(x):
    x+=15
    print(x)
y=3
test(y)
print(y)
```

مقدار بازگشتی تابع

یک تابع **یا هیچ مقدار بازگشتی ندارد، و یا تنها یک مقدار بازگشتی دارد.** مقدار بازگشتی چیست؟!
در واقعی مقداری است که پس از فراخوانی یک تابع به عنوان مقدار آن تابع ارزیابی می‌شود.

برای مثال، قبلا تابع len از توابعی که خود پایتون برایمان پیاده سازی کرده را دیده بودیم:

```
L = [2, 5, 3] # یک لیست همگن با ۳ عنصر از اندیس ۰ تا ۲ #
#-------------------------------------------------------
len(L)  # ارزیابی این عبارت عدد ۳ را نتیجه می‌دهد. #
```

در واقع تابع پس از فراخوانی مقداری بر می‌گرداند. این مقدار را میتوان به یک متغیر دیگر انتساب داد و یا همان جا آن را چاپ کرد:

```
a = len(L)  ✓
```

یا

```
print(len(L))  ✓
```

حال برای توابعی که خودمان تعریف می‌کنیم مقدار بازگشتی را با کلمه‌ی کلیدی return مشخص می‌کنیم.

مثال) برنامه‌ای بنویسید سه عدد به عنوان ورودی گرفته و جمع این سه عدد را به عنوان **مقدار بازگشتی** تابع برگرداند:

```
def sum_3num(a,b,c):
    return a+b+c
```

مثال) برنامه‌ای بنویسید سه عدد به عنوان ورودی گرفته و جمع این سه عدد را چاپ کند، این تابع هیچ مقدار بازگشتی نداشته باشد:

```
def sum_3num(a,b,c):
    print(a+b+c)
```

مثال) برنامه‌ای بنویسید سه عدد به عنوان ورودی گرفته و جمع این سه عدد را هم چاپ کند، هم به عنوان مقدار بازگشتی برگرداند:

```
def sum_3num(a,b,c):
    d = a+b+c
    print(d)
    return d
```

نکته: هر وقت یک تابع به دستور `return` رسید از تابع خارج شده و دیگر دستورات بعد از آن اجرا نخواهد شد. برای مثال کد زیر هیچ وقت دستور `print`ی که در تابع نوشته شده را اجرا نخواهد کرد:

```
def sum_3num(a,b,c):
    d = a+b+c
    return d
    print(d)
```

مثال) دو تابع زیر را در نظر بگیرید:

```
def test1(a):
    return a*5
def test2(a):
    print(a*5)
```

حال به نتیجه‌ی فراخوانی های زیر دقت کنید:

```
a= test1(6) # انتساب عدد ۳۰ به متغیر a
print(test1(3)) # چاپ عدد ۱۵ روی صفحه
test1(4) #نتیجه‌ی ارزیابی این تابع ۲۰ می‌شود، اما مقداری روی صفحه چاپ نمی‌شود
```

a= **test2(6)** *#عدد ۳۰ روی صفحه چاپ می‌شود*

در این مثال چون تابع test2 مقدار بازگشتی ندارد، مقدار با معنی به a نسبت داده نمی‌شود، در نتیجه a از نوع NoneType می‌شود.

print(**test2(3)**) *# print **15** and **None***

در نتیجه‌ی اجرای تابع مقدار ۱۵ روی صفحه چاپ می‌شود. اما خود تابع ارزیابی نمی‌شود و print تابعی که خروجی ندارد معنی ندارد! در نتیجه مقدار None چاپ می‌شود. همواره نتیجه‌ی چاپ متغیرهایی از نوع NonType مقدار None خواهد بود.

test2(4) *#عدد ۲۰ روی صفحه چاپ می‌گردد.*

مثال) تابعی بنویسید که سه عدد به عنوان وردی گرفته و میانگین این اعداد را به عنوان خروجی تابع (مقدار بازگشتی تابع) برگرداند.

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

مثال) تابعی بنویسید که سه عدد به عنوان وردی گرفته و عدد بزگتر را روی صفحه چاپ کرده و میانگین این اعداد را به عنوان خروجی تابع (مقدار بازگشتی تابع) برگرداند.

```
def print_largest_return_average(a,b,c):
    if a>b and a>c:
        print(a)
    elif b>c:
        print(b)
    else:
        print(c)
    d = a+b+c
    return d/3
```

مثال) تابعی به نام find_list بنویسید که دو ورودی داشته باشد، یک لیست و یک متغیر از هر نوع. این تابع باید در صورتی که آن متغیر در لیست وجود داشته باشد مقدار True و در غیر این صورت مقدار False برگرداند:

✓ در جدول زیر این مثال با ۲ روش مختلف حل شده است.

```
def find_list(l,n):
    f=0
    for e in l:
        if e==n:
            f=1
    if f==1:
        return True
    else:
        return False
```

```
def find_list(l,n):
    for e in l:
        if e==n:
            return True
    return False
```

سمت چپ این تابع با ایده‌ی تعریف یک flag مطابق برنامه‌های پیشین که بدون تابع حل کرده‌اید نوشته شده است.
اما با توجه به این نکته که هر وقت یک تابع به دستور return برسد ادامه‌ی دستورات درون آن تابع اجرا نمی‌شود، می‌توان بدون در نظر گرفتن flag برنامه را مطابق سمت راست نوشت. بدین صورت بلافاصله پس از این‌که برنامه عنصر را پیدا می‌کند مقدار True را return کرده و از تابع خارج میشود، اما اگر در جریان حلقه هیچ وقت e==n نشد، در خط آخر تابع مقدار False را return می‌کند.

سوال) با تابعی `find_list` که در مثال قبل نوشتیم، برنامه ای بنویسید که یک لیست از مقادیر از کابر گرفته و سپس یک مقدار برای جست و جو در لیست بگیرد. اگر آن مقدار در لیست بود عبارت `find` در غیر این صورت عبارت `not found` را چاپ کند.

```
def find_list(l,n):
    for e in l:
        if e==n:
            return True
    return False

n= int(input("how many items do you have?"))
i=0
mylist=[]
while (i<n):
    m=input("Enter a value:")
    mylist.append(m)
    i+=1
num= input("Enter a value to search in a list:")

if find_list(mylist,num):
    print("find")
else:
    print("not found")
```

نکته: نیازی به تعریف این تابع نیست. پایتون برای سادگی کار و کاهش حجم کد با استفاده از دستور `v in list` و `v not in list` این کار را برایمان انجام می‌دهد.

```
L = [1, 5, True, [2,3], 'srttu']

1 in L # ارزیابی می‌شود True

7 in L # ارزیابی می‌شود False

2 in L # ارزیابی می‌شود False

[2,3] in L # ارزیابی می‌شود True

1 not in L # ارزیابی می‌شود False

7 not in L # ارزیابی می‌شود True
```

مثال) به جای تابع find_list از in یا not in استفاده کنید و مثال قبل را مجددا حل کنید:

```
n= int(input("how many items do you have?"))
i=0
mylist=[]
while (i<n):
    m=input("Enter a value:")
    mylist.append(m)
    i+=1
num= input("Enter a value to search in a list:")

if num in mylist:
    print("find")
else:
    print("not found")
```

می‌توان in و not in را علاوه بر لیست ها برای رشته ها نیز استفاده کرد.

مثال) برنامه ای بنویسید که یک نام از کاربر گرفته، و اگر این نام حاوی حرف a بود اعلام کند has a :

```
name = input('Enter a name: ')
if 'a' in name:

    print('has a')
```

فراخوانی توابع تو در تو

می‌توانیم از توابعی که قبلا نوشته ایم در تعریف توابع جدید استفاده کنیم.

برای مثال فرض کنید یک تابع به نام f داریم که دو عدد به عنوان ورودی گرفته و مقدار کوچکتر را بازگرداند:

```
def f(a,b):
    if a>b:
        return b
    else:
        return a
```

حال فرض کنید بخواهیم تابعی به نام g بنویسم که از تابع قبلی استفاده کند و چهار عدد به عنوان ورودی گرفته و کوچکترین را چاپ کند؛ خواهیم داشت:

```
def g(a,b,c,d):
     v1 = f(a,b)
     v2 = f(c,d)
     return f(v1,v2)
```