لمس پیتون

تابع و بازگشت

گزارشی از محسن هوشمند و جواد اصغری

تابع‌ها

تاکنون تابع‌های داخلی پایتون معرفی شد و یا از تابع‌های کتابخانه استاندارد پایتون استفاده شد. اما در برنامه‌نویسی شایع است که کاری پیش آید که با تابع‌های موجود انجام نشوند. در نتیجه لازم می‌شود که تابعی را برای کار و وظیفهٔ مدنظر نوشت و ترتیب داد. دلایل استفاده از تابع یا تقسیم برنامه به تابع‌ها عبارتند از

- خواندن و خطایابی راحت‌تر
- کوچکتر کردن برنامه با حذف کدهای تکراری
- صرفا نیاز به تغییر در یک تابع و عدم نیاز به تغیر قسمت‌ها و بخش‌های مختلف برنامه
- امکان رفع خطای هر یک از توابع در هر مقطع زمانی و سپس سرهم‌کردن همگی آنها
- امکان بازاستفاده از آن در دیگر برنامه‌ها

بدون فوت وقت نخستین تابع جدید را تعریف می‌کنیم. تابع زیر توان سوم عدد ورودی را حساب می‌کند.

```
def tavan_se(adad):
  """Mohasebeye tavvan sevvome adad"""
  return adad * adad * adad

tavan_se(3)
27
```

```
tavan_se(1.2)
1.728
```

```
tavan_se('se')
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [4], in <cell line: 1>()
----> 1 tavan_se('se')

Input In [1], in tavan_se(adad)
      1 def tavan_se(adad):
      2     """MOhasebeye tavvan sevvome adad"""
----> 3     return adad * adad * adad

TypeError: can't multiply sequence by non-int of type 'str'
```

تابع تعریف شده برای کار با عدد صحیح و اعشاری تعریف شده است و با ورودی نوع «رشته» خطا می‌دهد.

چنان که از مثال برمی‌آید تعریف تابع با کلیدواژهٔ def آغاز می‌شود سپس نام تابع و پس از آن در پرانتز تعداد ورودی و در پایان خط دونقطه قرار می‌گیرد یا def *Functioname(arguments):* نام رشته‌ای از حروف و عدد و زیرخط است که بدون فاصله می‌آید و با عدد نیز آغاز نمی‌شود. تابع دارای یک یا چند آرگومان ورودی است. اگر چند آرگومان ورودی نیاز داشته باشد بین آنها با کاما مشخص می‌شود. بدنه تابع دارای تورفتگی است. در خط دوم بهتر است و پیشنهاد توضیح کار کلی و هدف تابع در سه‌نقل قول ذکر شود. در صورتی که قرار باشد مقداری را برگرداند در انتها با دستور *return* کار مذکور انجام می‌شود.

در صورتی که قصد خواندن سند تابعی را داشته باشیم از علامت پرسش ? استفاده می‌شود.

```
tavan_se?
```

```
Signature: tavan_se(adad)
Docstring: Mohasebeye tavvan sevvome adad
File:      c:\users\user1\appdata\local\temp\ipykernel_10220\220731020.py
Type:      function
```

نام تابع و علامت پرسش اطلاعات زیر را از تابع نمایش می‌دهد

- نام و فهرست پارامترهای ورودی تابع که معروف به امضای تابع است
- سند-رشته تابع
- نام فایلی که تابع در آن تعریف شده است.
- نوع مدخل

اگر از دو علامت پرسش پیاپی استفاده شود، در صورت موجود بودن کد آن نیز نمایش داده می‌شود.

```
tavan_se??
```

```
Signature: tavan_se(adad)
Source:
def tavan_se(adad):
    """Mohasebeye tavvan sevvome adad"""
    return adad * adad * adad
File:      c:\users\user1\appdata\local\temp\ipykernel_10220\220731020.py
Type:      function
```

برگرداندن مقدار: اگر تابعی خروجی داشت نیاز است مقدار تولیدی آن برگردانده شود. جهت برگرداندن مقدار از تابع از دستور return استفاده می‌کنیم. توابع معمولا نتیجه‌ای را برمی‌گرداند و از دستور return در انتهای اجرا جهت دستیابی به چنین مهمی بهره می‌برند. توابعی دیگر وجود دارند که صرفا عملی را انجام می‌دهند و خروجی جدیدی تولید نمی‌کنند به آنها توابع تهی می‌گوییم. مثال زیر دو بار جمله ورودی را چاپ می‌کند و خروجی خاصی (ایجاد متغیری یا داده‌ای جدید) را در برندارد.

```
def do_bar_chap(jomle):
    """do bar chap reshteye vorodi"""
    print(jomle)
    print(jomle)
do_bar_chap('AchiMachiLaTarachi ...')
```

```
AchiMachiLaTarachi...
AchiMachiLaTarachi...
```

به نحوه استفاده از تعریف تابع بالا در مثال زیر توجه کنید.

```
b = do_bar_chap('AchiMachiLaTarachi ...')
print(b)
AchiMachiLaTarachi ...
AchiMachiLaTarachi ...
None
```

همان‌طور که نتیجهٔ مثال بالا نمایش داد ، وقتی برگردان یا «return» در تعریف تابع استفاده نشده باشد، چیزی به متغیر «b» تخصیص نخواهد یافت.

مثال زیر رشتهٔ ورودی را به تعداد دلخواه n دفعه چاپ خواهد کرد.

```
def n_bar_chap(jomle, n):
    """be tedat vorodi n, n bar chap reshteye vorodi"""
    for i in range(n):
        print(jomle)

n_bar_chap('Baz sho konjed!', 4)
Baz sho konjed!
Baz sho konjed!
Baz sho konjed!
Baz sho konjed!
```

مثال را می‌توان بدون استفاده حلقه for نوشت.

```
def n_bar_chap2(jomle, n):
    """be tedat vorodi n, n bar chap reshteye vorodi"""
    print((jomle + '\n') * n)

n_bar_chap2('Varzesh kon!', 5)
Varzesh kon!
Varzesh kon!
Varzesh kon!
Varzesh kon!
Varzesh kon!
```

لازم به ذکر است عدم استفاده صریح از حلقه به معنای عدم نیاز به استفاده از دستورات تکرار نیست. در واقع تکرار را با استفاده از ضرب چند باره رشته بازنویسی کردیم.

اگر در تابع متغیری تعریف شود «محلی» محسوب می‌شود. به بیان دیگر، صرفا در دورن تابع باقی خواهد ماند و پس از اجرای تابع از بین خواهد رفت. مثال

```
def ettesal_dotayee(bakhsh1, bakhsh2):
    ettesal = bakhsh1 + ' ' + bakhsh2
    return ettesal
```

تابع بالا دو ورودی را دریافت و آنها را متصل و جمله حاصل را دو بار چاپ می‌کند.

```
nam = 'Iraj '
shohrat = 'Afshar'
etesal_dotaye(nam, shohrat)
Iraj Afshar
```

با اتمام اجرای تابع etesal_dotaye ، متغیر etesal نابود می‌شود.

```
print(etesal)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-21-34599fba884e> in <module>
----> 1 print(etesal)

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

مثال‌های قبلی نشان می‌دهند لازم نیست خود مقدار در فراخوانی تابع در ارگومان‌های تابع قرار گیرد و صرفا با تعریف متغیر و قرار دادن متغیر تابع به مقدار دست خواهد یافت (امکان ورودی متغیر به جای آرگومان).

مثال زیر تابعی تعریف می‌شود که ورودی‌های آن طول و عرض و خروجی آن مساحت مستطیل متناظر است.

```
def masaht_mostatil(tool, arz):
    masahat = tool * arz
    return masahat

m = masaht_mostatil(5, 4)
print('masahat barabar ba ', m)
```

```
masahat barabar ba  20
```

مثال زیر دو خروجی دارد. هم مساحت و هم محیط را محاسبه می‌کند.

```
def masaht_mohit_mostatil(tool, arz):
    masahat = tool * arz
    mohit = (tool + arz ) * 2
    return masahat, mohit


t = 5
a = 4
ma, mo = masaht_mohit_mostatil(t, a)
print(ma)
print(mo)
```

```
20
18
```

خروجی مثال بالا، ساختار داده‌ای معروف به «چندتایی» است که در فصل بعد آن را بیشتر بررسی خواهیم کرد.

مثال زیر مساحت دایره را حساب می‌کند. ورودی آن اندازهٔ شعاع است.

```
def masahat_dayere(shoae):
    m = math.pi * shoae ** 2
    return m
```

می‌توان از تعریف متغیر محلی نیز اجتناب کرد و تابع را به صورتی موجزتر تعریف کرد.

```
def masahat_dayere2(shoae):
    return math.pi * shoae ** 2
```

```
import math
masahat_dayere2(5)
78.53981633974483
```

مثال زیر نیز دارای برگردان از نوع چندتایی است و قطر و محیط و مساحت دایره را با ورودی شعاع حساب می‌کند.

```
def masaht_mohit_dayere2(s):
   if s < 0:
      print('mqdare shoae manfi ast!')
   else:
      qo = s * 2
      ma = math.pi * s ** 2
      mo = math.pi * 2 * s
      return qo, ma, mo


import math
sh = -1
masaht_mohit_dayere2(sh)
```

مثال قبلی شامل چند نکتهٔ مهم دیگر است. چون شعاع منفی تعریف نشده است، پس در ابتدا منفی بودن آن بررسی می‌شود و در صورت احراز پیامی خطا نمایش داده خواهد شد و از اجرای بقیهٔ کد صرف‌نظر می‌شود. در صورت نامنفی بودن بقیه کد اجرا خواهد شد. جهت ایجاد مقدار پی از پیمانهٔ ریاضی «math» کتابخانه استانده پایتون استفاده می‌شود.

حال تابع را برای چندین مقدار متفاوت اجرا می‌کنیم. قطعه کد زیر را ببینید.

```
import math
for i in range(-4, 11, 2):
   print(' shoae : ', i, ' be tartib qotr, masahat, mohit: ', end =' ')
   print(masaht_mohit_dayere3(i))
 shoae :  -4  be tartib qotr, masahat, mohit:  'mqdare shoae manfi ast!
 shoae :  -2  be tartib qotr, masahat, mohit:  'mqdare shoae manfi ast!
 shoae :  0  be tartib qotr, masahat, mohit:  (0, 0.0, 0.0)
 shoae :  2  be tartib qotr, masahat, mohit:  (4, 12.566370614359172,
12.566370614359172)
 shoae :  4  be tartib qotr, masahat, mohit:  (8, 50.26548245743669,
25.132741228718345)
 shoae :  6  be tartib qotr, masahat, mohit:  (12, 113.09733552923255,
37.69911184307752)
 shoae :  8  be tartib qotr, masahat, mohit:  (16, 201.06192982974676,
50.26548245743669)
 shoae :  10  be tartib qotr, masahat, mohit:  (20, 314.1592653589793,
62.83185307179586)
```

- تمرین- مقادیر را با دقت دو رقم اعشار بنویسید و همین‌طور نمایش را با رشته f منظم کنید.

تابعی ممکن است بسته به احراز شرط یا شرایطی چند نوع مقدار متفاوت را برگرداند. مثال زیر قدر مطلق را حساب می‌کند و با توجه به مقدار شرط دو برگردان متفاوت دارد.

```
def qadr_motlaq(x):
```

```
    if x < 0:
        return -x
    else:
        return x

qadr_motlaq(-5.5)
5.5
```

توسعه تدریجی: در هنگام نوشتن تابع و برنامه‌نویسی توصیه می‌شود که به صورت تدریجی کد تکمیل شود. کار مزبور جهت پیشبرد بهینه و عاری از خطای کد توصیه می‌شود تا در هر مرحله بخشی از کد تکمیل شود. به عنوان مثال، دو نقطه در مختصات دو بعدی داده می‌شود تا فاصله اقلیدسی آنها محاسبه شود. مثال محاسبه فاصله بین دو نقطه را در ابتدا و در قدم اول به صورت زیر تعریف می‌کنیم.

```
def fasele(x1, y1, x2, y2):
    return 0.0
```

در قدم اول نشان داده‌ایم ورودی‌ها چیست و تابع حتما خروجی دارد. در قدم بعد می‌توان اختلاف هر راستا را متمایز از دیگری محاسبه کرد.

```
def fasele(x1, y1, x2, y2):
    dx = x1 - x2
    dy = y1 - y2
    print('dx = ', dx)
    print('dy = ', dy)
    return 0.0
```

در گام بعدی جمع مربع اختلاف در راستای هر محور را محاسبه می‌کنیم.

```
def fasele(x1, y1, x2, y2):
    dx = x1 - x2
    dy = y1 - y2
    f_morabae = dx ** 2 + dx ** 2
    return 0.0
```

در گام نهایی نحوهٔ محاسبهٔ فاصله را با فراخوانی تابع جذر از کتابخانه نهایی می‌کنیم. همچنین برگردان را برابر با مقدار فاصلهٔ محاسبه شده قرار می‌دهیم و بدین نحو فرایند نوشتن تدریجی را نهایی و تکمیل می‌کنیم.

```
def fasele(x1, y1, x2, y2):
    dx = x1 - x2
    dy = y1 - y2
    f_morabae = dx ** 2 + dy ** 2
    f = math.sqrt(f_morabae)
    return f
```

حال کد آمادهٔ اجرا است.

```
import math
fasele(4, 2, 5, 5)
1.4142135623730951
```

ترکیب: تاکنون بر خواننده روشن شده است که می‌توان تابعی را در تابعی دیگر فراخوانی کرد. به عنوان مثال در مثال زیر مساحت دایره را با دریافت مختصات مرکز دایره و نقطه‌ای روی محیط دایره بدست می‌آوریم. در ابتدا لازم است که اندازه شعاع بر اساس مختصات دو نقطه محاسبه شود که جهت اجرای وظیفهٔ مذکور «تابع محاسبه فاصله» را در «تابع محاسبه مساحت» فرا می‌خوانیم. در نهایت مقدار مساحت را حساب می‌کنیم.

```
def masahat_dayere3(x1, y1, x2, y2):
   shoae = fasele(x1, y1, x2, y2)
   return math.pi * shoae ** 2
```

```
import math
masahat_dayere3(0, 0, 1, 0)
3.141592653589793
```

- تمرین – می‌توان مربعی را با مختصات نقطه بالا-سمت چپ و طول ضلع تعیین نمود. تابعی بنویسید که با دریافت مختصات نقطه بالا-سمت چپ و اندازه ضلع دیگر نقاط و اندازه قطر مربع را حساب کند.

ممکن است تابع بسته به نیاز مسئله دارای چند پارامتر ورودی باشد. در مثال زیر، تابع سه مقدار را می‌گیرد و بیش آنها را برمی‌گرداند.

```
def bish(mqdar1, mqdar2, mqdar3):
   """yaftan o bargardandan mqdar bishine"""
   mqdar_bish = mqdar1
   if mqdar2 > mqdar_bish:
      mqdar_bish = mqdar2
   if mqdar3 > mqdar_bish:
      mqdar_bish = mqdar3
   return mqdar_bish

bish(10, 9, -10)
10

bish(1.0, 9.5, 4.30)
9.5

bish('sabz', 'sepid', 'sorkh')
'sorkh'
```

توجه شود که ترتیب الفبائی در نمایش مهم است و نه طول رشته‌ها:

```
bish_se_adad('sabzzzzzzzzzzzzzzzzzzzzz', 'sepid', 'sorkh')
'sorkh'
```

در صورت فراخوانی نوع مقدارهای متفاوت اجرا با خطا مواجه می‌شود.

```
bish('sabz', 10, 4.3)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
```

```
Input In [35], in <cell line: 1>()
----> 1 bish('sabz', 10, 4.3)

Input In [30], in bish(mqdar1, mqdar2, mqdar3)
      2 """yaftan o bargardandan mqdar bishine"""
      3 mqdar_bish = mqdar1
----> 4 if mqdar2 > mqdar_bish:
      5     mqdar_bish = mqdar2
      6 if mqdar3 > mqdar_bish:

TypeError: '>' not supported between instances of 'int' and 'str'
```

پس در مقایسهٔ انواع مختلف داده تابع خطا برمی‌گرداند

لازم به ذکر است که پایتون دارای توابع داخلی کم و بیش است.

```
max(10, 9, -10, 12, 45, 1)
45
```

```
min('fil', 'shir', 'moosh', 'palang')
'fil'
```

البته می‌توان فیل را به جایگاه خود برگرداند!

```
min('pil', 'shir', 'moosh', 'palang')
'moosh'
```

می‌توان برای تابع مقادیر پیش‌فرض تعریف کرد. به بیان دیگر، در صورت عدم ورودی، تابع خودبخود مقادیر پیش‌فرض را جانشانی خواهد کرد.

```
def masahat_mostatil(tol = 2, arz = 4):
  """bargardandane masahat mostatil"""
  return tol * arz
masahat_mostatil
<function __main__.masahat_mostatil(tol=2, arz=4)>
```

اجرای با برگرداندن مقدار

```
masahat_mostatil()
8
masahat_mostatil(12, 9)
108
```

ممکن است فهرست آرگومان‌های از قبل مشخص نباشد و تعداد آنها متغیر باشد. می‌توان از عملگر * استفاده کرد. تابع زیر میانگین‌گیری با تعداد دلخواه ورودی را تعریف می‌کند.

```
def miangin(*gozare):
  return sum(gozare) / len(gozare)
miangin(5, 10)
7.5
```

نمونه اجرای دیگر

```
miangin(5, 10, 15)
10.0
```

و نمونه دیگر:

```
miangin(5, 10, 15, 20)
12.5
```

عملگر * را می‌توان با هر شی چندمقداری بکار برد.

```
nomarat = [17, 12, 20, 15, 18, 14]
miangin(*nomarat)
16.0
```

روش یا متد در پایتون تابعی است که متعلق به یک شیء است که می‌توان آن را به صورت زیر فراخواند.

```
Nam_Shei.Nam_ravash(vorodiha)
```

مثلا متغیرهای رشته دارای چند روش متعلق به خود هستند. مثال زیر روش‌های کوچک‌سازی حروف را نشان می‌دهد.

```
r = 'Salam'
r.lower()
'salam'
```

مثال زیر روش‌های بزرگ‌سازی حروف را نشان می‌دهد.

```
r.upper()
'SALAM'
```

```
r
'Salam'
```

حوزه محلی و سراسری متغیرها

```
x = 7

def dastresi_sarasari():
   print('chap x ba dastresi sarasari', x)

dastresi_sarasari()
chap x ba dastresi sarasari 7
```

اما متغیر سراسری را نمی‌توان داخل تابع تغییر داد. در واقع داخل تابع یک متغیر محلی جدید ایجاد می‌کند.

```
def azmayesh_taqir_m_sarasari():
   x = 3.5
   print('chpa x az daron', x)

azmayesh_taqir_m_sarasari()
chpa x az daron 3.5
```

```
x
7
```

کتابخانه استانداۀ پایتون

تقسیم تابع‌ها و کلاس‌ها بر اساس شباهت عملکردی رخ می‌دهد و آنها در ماژول‌ها یا پیمانه‌ها قرار می‌گیرند. پیمانه فایلی است که دارای مجموعه‌ای از توابع مرتبط با هم است. چند پیمانۀ مرتبط تشکیل یک بسته یا package را می‌دهند. کتابخانه استانداۀ پایتون کانون اصلی زبان پایتون است، و بسته‌ها و پیمانه‌هایش طیف گسترده‌ای از ابزارها را مانند کار با فایل‌های ورودی/خروجی، بنیادهای ریاضی فراهم می‌کند. چند پیمانۀ معروف آن در جدول زیر آمده است.

از پیمانه‌های مهم پایتون math است که تابع‌های ریاضی را پشتیبانی می‌کند. پیتون دارای بخش‌های ریاضی است که توابع ریاضی را در دسترس قرار می‌دهد. در صورت نیاز به تابعی خاص می‌توان آن را به صورت زیر فراخواند.

دستور import با آوردن کلیدواژه مزبور و سپس نام پیمانه کار می‌کند. پس به کمک نام پیمانه و نام تابع بعد نقطه به تاب مدنظر دسترسی می‌یابیم. فراخوانی پیمانۀ مذکور به صورت زیر است..

```
import math
```

اما گاهی اوقات فقط بخشی از پیمانه لازم است می‌توان از from… import بهره برد. در حالت مزبور نیازی به نقطه نیز نیست.

```
from math import sin
x = 3.14
y = sin ( x )
print ( y )
```

اگر تعدادی محدود تابع نیاز داشته باشیم می‌توان به صورت زیر عمل کرد:

```
from math import sin , cos
x = 3.14
y = sin ( x )
print ( y )

y = cos ( x )
print ( y )
```

یا

```
from math import ceil, floor

ceil(10.4)
11
floor(10.4)
11
```

امکان نام‌گذاری پیمانه‌ها نیز وجود دارد. مثال زیر نحوه انجام کار را نشان می‌دهد.

```
import statistics as stats

nomarat = [17, 12, 20, 15, 18, 14]



stats.mean(nomarat)
16
```

اگر نیاز به تعداد زیادی از تابع‌ها داشته باشیم می‌توان به صورت زیر عمل کرد.

```
from math import *
x = 3.14
y = sin ( x )
print ( y )

y = cos ( x )
print ( y )
```

با این دستور

```
math
<module 'math' (built-in)>
```

جهت استفاده از توابع پیمانه، نیاز است که از نمادگذاری نقطه استفاده کرد و با گزارهٔ ورود آن را فراخواند.

```
d= 10*math.log10(۱۰۰۰)
print(d)
```

```
math.sqrt(2) / 2.0
print(math.pi)
print(math.log2(8))
math.fabs(-13)
```

دستور بالا قدر مطلق را با نوع اعشاری برمی‌گرداند.

جدول زیر چند تابع مهم پیمانه ریاضی را نشان می‌دهد.

تابع	توضیح
ceil(x)	گردسازی به کوچکترین عدد صحیح بزرگتر از x
floor(x)	گردسازی به بزرگترین عدد صحیح کوچکتر از x
pow(x,y)	x^y
exp(x)	e^x
sqrt(x)	$\sqrt{x}$
fabs(x)	قدر مطلق اعشاری x
fmod(x,y)	باقیمانده اعضاری تقسیم x بر y
sin(x)	x مقداری رادیانی
cos(x)	x مقداری رادیانی
tan(x)	x مقداری رادیانی

asin(x)	سینوس معکوس ورودی
acos(x)	کسینوس معکوس ورودی
atan(x)	تانژانت معکوس ورودی

مثال

```
from math import *
print(ceil(1.6))
print(floor(1.6))
print(pow(2, 10))
print(exp(3))
print(sqrt(4))
print(fmod(9,5))
print(fmod(9.8, 4.0))
```

```
2
1
1024.0
20.085536923187668
2.0
4.0
1.8000000000000007
```

```
In [3]: math.<Tab>
        acos()      atan()      copysign()  e           expm1()
        acosh()     atan2()     cos()       erf()       fabs()
        asin()      atanh()     cosh()      erfc()      factorial() >
        asinh()     ceil()      degrees()   exp()       floor()
```

در صورتی که بخواهیم از توابع داخلی پیمانه‌ای مطلع شویم کافی است نام آن را به نقطه و سپس کلید «تب» را کلیک کنیم:

```
math.<Tab>
```

جهت اطلاع از توابع پیمانه می‌توان شبیه مثال زیر عمل کرد.

```
math.fabs?
```

```
Signature: math.fabs(x, /)
Docstring: Return the absolute value of the float x.
Type:      builtin_function_or_method
```

مثال زیر نحوه تبدیل درجه به رادیان را نشان می‌دهد. عدد π را از کتابخانه math فرامی‌خوانیم.

```
import math as mt

def tabdil_daraje_be_radian(daraje):
    radian = mt.pi * daraje / 180
    return radian
tabdil_daraje_be_radian(60)
```

```
1.0471975511965976
```

ترکیب: تقریبا هر کجا که بتوان مقداری وارد کرد می‌توان عبارتی دلخواه را جای آن گذاشت.

```
x = math.sin(degrees / 360.0 * 2 * math.pi)
x = math.exp(math.log(x+1))
```

البته می‌توان به صورت دیگری نیز پیمانه را فراخواند.

```
import math as mt
x = 3.14
y = mt.sin ( x )
print ( y )
```

فراخوانی اخیر از این لحاظ اهمیت دارد که ممکن است پیمانه‌های متفاوت تابع‌های یکسانی داشته باشند. مثال زیر را ببینید.

```
import math as mt
import numpy as np

x = 3.14
y = mt.sin ( x )
print ( y )

y = np.sin ( x )
print ( y )
```

همان‌طور که بالاتر ذکر شد کتابخانه استانداۀ پایتون ابزارهای و تسهیلات زیادی در قالب پیمانه‌ها فراهم می‌کند. اما لازم است قبل از اجرا فراخوانی و نصب شوند. سپس، در متن کد وارد شوند و فراخوانی شوند و در نهایت از تابع‌هایش استفاده کرد. چند بستۀ مهم عبارتند از:

- نامپای NumPy: بسته‌بنیادی رایانش علمی با پیتون است. در دیگر بخش‌ها و فصل‌های آینده بیشتر معرفی خواهد شد.
- سای‌پای SciPy: بسته‌ای متن‌باز ازکتابخانه پیتون است که جهت رایانش علمی و محاسبات فنی استفاده می‌شود. سای‌پای داری پیمانه‌هایی جهت بهینه‌ازی،جبرخطی، انتگرال‌گیری، درون‌یابی، توابع خاص، تبدیل سریع فوریه، پردازش سیگنال و تصویر، تسهیل‌گرهای حل معالات دیفرانسیلی معمولی، و کارهای دیگر معمول در علم و مهندسی است.
- مت‌پلات‌لیب Matplotlib: جهت رسم‌های پیتونی استفاده می‌شود.

اعداد تصادفی

اعداد تصادفی نقشی مهم در شبیه‌سازی‌های عددی و کارهای رایانه‌ای دارند. زبان‌های برنامه‌نویسی جهت برآورده کردن نیاز مزبور ابزارهای تولید اعداد شبه‌تصادفی را در اختیار می‌گذارند. پایتون نیز از این امر مستثنی نیست و پیمانه‌ای به نام random را برای این مهم در کتابخانه استانده پایتون فراهم ساخته است. مثال زیر ده بار پرتاب تاس را مدل می‌کند.

```
import random

for tas in range(10):
   print(random.randrange(1, 7), end=' ')
6 5 6 5 4 6 5 5 4 3
```

در قدم اول پیمانه random فراخوانی شد تا بتوان از امکانات آن بهره برد. تابع randrange دو آرگومان ورودی دارد و عددی تصادفی بین دو مقدار ورودی (به جز خود عدد دوم) تولید می‌کند. اگر حلقه بالا دوباره اجرا شود ترتیب دیگری از مقادیر به دست خواهد داد.

```
3 1 5 6 3 5 6 5 4 6
```

ممکن است سوال پیش آید که آیا تابع مذکور در اساس اعداد تصادفی تولید می‌کند؟ در مثال زیر سعی می‌کنیم چند میلیون بار تاس را پرتاب کنیم و سپس فراوانی هر برآمد را بشماریم.

```
import random

# Shomaresh farvani har vajhe tas
faravani1 = 0
faravani2 = 0
faravani3 = 0
faravani4 = 0
faravani5 = 0
faravani6 = 0

#partab milioni tas
tedad_partab = 6_000_000
for tas in range(tedad_partab):
   vajh = random.randrange(1, 7)

   # afzayesh tedad farvani vajhs motanzer baramade partab
   if vajh == 1:
      faravani1 += 1
   elif vajh == 2:
      faravani2 += 1
   elif vajh == 3:
      faravani3 += 1
   elif vajh == 4:
      faravani4 += 1
   elif vajh == 5:
      faravani5 += 1
   else:
      faravani6 += 1
# mohasebe nesbat tekrare har vajh
nesbat1 = faravani1 / tedad_partab
nesbat2 = faravani2 / tedad_partab
nesbat3 = faravani3 / tedad_partab
nesbat4 = faravani4 / tedad_partab
nesbat5 = faravani5 / tedad_partab
nesbat6 = faravani6 / tedad_partab

# Chap natayej
print(f'Vajh{"Farvani":>13}{"Nesbat":>13}')
print(f'{1:>4}{faravani1:>13}{nesbat1:>13.4f}')
print(f'{2:>4}{faravani2:>13}{nesbat2:>13.4f}')
print(f'{3:>4}{faravani3:>13}{nesbat3:>13.4f}')
print(f'{4:>4}{faravani4:>13}{nesbat4:>13.4f}')
print(f'{5:>4}{faravani5:>13}{nesbat5:>13.4f}')
print(f'{6:>4}{faravani6:>13}{nesbat6:>13.4f}')
```

```
Vajh      Farvani       Nesbat
   1      1000827         0.17
   2       999870         0.17
   3      1000924         0.17
```

```
   4       998887         0.17
   5       999183         0.17
   6      1000309         0.17
```

یا با افزایش نسبت اعشاری جهت نمایش

```
Vajh       Farvani       Nesbat
   1        998213       0.1664
   2        998513       0.1664
   3       1000088       0.1667
   4       1000522       0.1668
   5       1002641       0.1671
   6       1000023       0.1667
```

همان‌طور که از مثال پیداست برای محاسبه فراوانی هر وجه تاس یک متغیر و جمعا شش متغیر تعریف شده است. همین‌طور برای محاسبهٔ نسبت نیز شش متغیر متفاوت به تعداد وجه‌ها تعریف شده است. اگر بتوان آنها را فشرده‌تر نوشت و در نمایش‌های چندمقداری قرار داده شوند، نه تنها نمایش آنها ساده‌تر بلکه استفاده از حلقه نیز ممکن می‌شود و برنامه را با تعداد خطوطی کمتر و اشتباه کمتر می‌توان نوشت. می‌توان از فهرست شش مقداری برای متغیرهای فراوانی و نسبت استفاده کرد. پس مثال را به صورت زیر بازنویسی می‌کنیم.

```
import random

# Shomaresh farvani har vajhe tas
faravani = [0] * 6

#partab milioni tas
tedad_partab = 6_000
for tas in range(tedad_partab):
   vajh = random.randrange(1, 7)
   # afzayesh tedad farvani vajhs motanzer baramade partab
   faravani[vajh-1] += 1

# Mohasebeye nesbat tekrare har vajh
nesbat = [f / tedad_partab for f in faravani]

# Chap natayej
print(f'Vajh{"Farvani":>13}{"Nesbat":>13}')
for i in range(1,7):
   print(f'{i:>4}{faravani[i-1]:>13}{nesbat[i-1]:>13.4f}')
Vajh       Farvani       Nesbat
   1       1000827         0.17
   2        999870         0.17
   3       1000924         0.17
   4        998887         0.17
   5        999183         0.17
   6       1000309         0.17
```

مثال بالا نمونه‌ای از استفاده از فهرست را برای مقادیر فراوانی و نسبت نشان داد. در فصل بعد بیشتر به این مطلب خواهیم پرداخت.

تابع randrange عددی شبه تصادفی است. در سازوکار داخلی خود از مفهومی به نام بذر استفاده می‌کند. در هر بار اجرا از بذری متفاوت استفاده می‌کند. گاهی اوقات لازم است که برنامه چند بار اجرا شود ولی اعداد تصادفی یکسانی برگرداند! جهت تامین منظور از تابع seed در پیمانهٔ random استفاده می‌شود.

```
import random

random.seed(32)

for tas in range(10):
   print(random.randrange(1, 7), end=' ')
1 2 2 3 6 2 4 1 6 1
```

```
random.seed(15)

for tas in range(10):
   print(random.randrange(1, 7), end=' ')
2 1 5 6 1 2 2 1 1 6
```

```
random.seed(32)

for tas in range(10):
   print(random.randrange(1, 7), end=' ')
1 2 2 3 6 2 4 1 6 1
```

در اینجا به خوانندهٔ مطلع صرفا اشاره‌ای می‌کنیم که پایتون همانند سی و سی++ اشاره‌گر را فراهم می‌کند.

```
x = 7
id(x)
2335861637552
```

پیمانهٔ آمار پیتون

پیتون ابزارهای آماری نیز در اختیار می‌نهد. چند آمار توصیفی را با استفاده از دستورات داخلی پیتون و یا کتابخانهٔ آمار بررسی می‌کنیم. آمارهای مذکور شامل میانگین، میانه، و نمایه هستند. سه معیار مزبور معروف به معیارهای گرایش مرکزی هستند. جهت اجرای مثال از چند تابع داخلی پایتون اعم از len و sum استفاده می‌کنیم. محاسبهٔ میانگین به صورت زیر است.

```
nomarat = [18, 19, 16, 14, 16]

sum(nomarat) / len(nomarat)
16.6
```

اما با استفاده از پیمانهٔ statistics می‌توان میانگین و سایر معیارها را محاسبه کرد. پس در ابتدا پیمانه را فرا می‌خوانیم.

```
import statistics

statistics.mean(nomarat)
16.6
```

```
statistics.median(nomarat)
16
```

```
statistics.mode(nomarat)
16
```

```
sorted(nomarat)
[14, 16, 16, 18, 19]
```

بازگشت و تابع برگشت‌دار

هر تابعی می‌تواند تابعی متفاوت را فرا بخواند. همچنین می‌تواند خود را فرابخواند! از نقاط قوت برنامه فراخوانی خود تابع در تابع است. در نتیجه تابع بازگشتیْ تابعی است که خود را فراخوانی کند. چنین اجرائی را «بازگشت» خوانند.

تابع شمارش برعکس را می‌توان به صورت زیر نوشت:

```
def shomaresh_makos(n):
   while n > 0:
      print(n)
      n = n - 1
   print('Atash')
shomaresh_makossh_makos(9)
```

اما می‌توان آن را به صورت بازگشتی نیز نوشت:

```
def shomaresh_makos_bazgasgti(n):
   if n <= 0 :
      print('Atash')
   else:
      print(n)
      shomaresh_makos_bazgasgti(n - 1)
shomaresh_makos_bazgasgti(9)
```

نسخه دیگر از تابع countdown

```
def donbale(n):
while n != 1:
   print(n)
   if n % 2 == 0:    # n is even
      n = n / 2
   else:             # n is odd
      n = n*3 + 1
```

مثال- تابعی که رشته‌ای را n بار چاپ کند.

```
def chap_n(s, n):
   if n <= 0:
      return
   print(s)
   chap_n(s, n-1)
```

```
chap_n('salam', 4)
salam
salam
salam
salam
```

مثال زیر نحوه محاسبه فاکتوریل را اجرا می‌کند.

$$0! = ۱$$

$$n! = n(n - ۱)!$$

```
def factorial(n):
    if n == 0:
        return 1
    else:
        bazgasht = factorial(n-1)
        result = n * bazgasht
        return result
```

قبل از ادامه مثالی دیگری از تابع بازگشتی لازم است اشاره شود که تابع‌ها نوع ورودی را بررسی نمی‌کنند. می‌توان وارسی نوع‌ها را انجام داد.

مثال زیر را اجرا کنید.

```
factorial(1.5)
```

خطائی شبیه زیر دریافت خواهد شد.

```
RuntimeError: Maximum recursion depth exceeded
```

استفاده از تابع پیتون وارسی نوع و برگرداندن گزارش مناسب حال است:

```
def factorial(n):
    if not isinstance(n, int):
        print('Factorial serfan baraye adad sahih tarif shode ast.')
        return None
    elif n < 0:
        print('Factorial baraye adad sahih namanfi tarif nashode ast.')
        return None
    elif n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

حال آرگومانی غیرمرتبط را وارد کنید. مثال‌های زیر چنین ورودی‌هائی هستند:

```
factorial(1.5)
Factorial serfan baraye adad sahih tarif shode ast.
```

```
factorial(-10)
Factorial baraye adad sahih namanfi tarif nashode ast.
```

```
factorial(0)
1
```

```
factorial(5)
120
```

عدد فیبوناتچی را به صورت زیر و بازگشتی حساب می‌کنیم:

```
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

for i in range(11):
    print('fib(', i, ')=', fibonacci(i))
```

برنامه بالا زمان زیادی را می‌گیرد و بسیاری از مقادیر را چندین بار محاسبه کرد. به سخن دیگر، تابع بازگشتی لزوما در همه‌جا دارای بهینگی زمان اجرا نیست. می‌توان تابع فیبوناچی را به صورت زیر نیز نوشت تا از مشکل حاصل از برنامه بازگشتی جلوگیری کرد.

```
def fibo(n):
    f = [0]*(n + 1)
#     print(len(f))
    f[0] = 1
    f[1] = 1
    for i in range(2, n+1):
        f[i] = f[i - 1] + f[i - 2]
    return f[n]

for i in range(11):
    print('fib(', i, ')=', fibonacci(i))
```

می‌توان از استفاده از فهرست نیز اجتناب کرد. پس برنامه فیبوناتچی را به صورت زیر بازنویسی می‌کنیم.

```
def fibo3(n):
    f_qabli = 1
    f_2qabli = 1

    if n == 0 or n == 1:
        return 1
    for i in range(2, n+1):
        f = f_qabli + f_2qabli
        f_2qabli = f_qabli
        f_qabli = f

    return f

for i in range(11):
    print('fib(', i, ')=', fibonacci(i)
```

بازگشت بی‌نهایت: در صورت نرسیدن به حالت پایه، بازگشت برای همیشه خواهد شد و برنامه هرگز پایان نخواهد یافت. چنین حالتی مشهور به بازگشت بی‌نهایت است. همیشه سعی می‌شود از چنین حالتی اجتناب کرد. مثال

```
def recurse():
        recurse()
```

وارسی نوع: گاهی اوقات تابع‌ها صرفا ورودی خاصی را می‌پذیرند. بنابراین، در صورتی که نوع داده غلط وارد شود یا نتیجه درستی نخواهد بود یا خطا صادر خواهد شد. می‌توان تدابیری برای جلوگیری از ورودی نادرست اتخاذ کرد. همچنین، پایتون امکاناتی را در اختیار قرار می‌دهد. در مثال زیر با دستوری می‌توان نوع داده را بررسی کرد. در وردی فاکتوریل داده باید حتما عدد صحیح نامنفی باشد و با استفاده از دستور داخلی پایتون آن را بررسی و بر اساس نتیجه وارسی کار متناسب اعم از گزارش خطا یا محاسبه مقدار ورودی درست انجام خواهد یافت.

رفع خطا و خطایابی: فاصله امکان ایجاد خطا دارد بنابراین در حین خطا بررسی کنید که بین تابع و ارگومان‌هایش فاصله‌ای نباشد. توجه به این نکته ضروری است که پیام خطا نمایشگر این هستند که کجا مشکل کشف شده است. ولی خطا ممکن است در جایی بالاتر در خطوط کد اتفاق افتاده باشد و لازم شود خطوط بالاتر بررسی شود. مهم‌ترین توانایی برنامه‌نویس رفع خطاست. گفته می‌شود برنامه‌نویسی با رفع خطا یکسان هستند. برنامه‌نویسی فرایند رفع خطای تا هنگام دستیابی به برنامه‌ای است که می‌خواهید. لینوکس سیستم عاملی که میلیون‌ها خط کد دارد اما در ابتدا برنامه‌یا جهت کاوش چیپ اینتل ۸۰۸۶ بود. از پروژه‌های اولیه لینوس تراولدوس برنامه‌ای بود که بین چاپ AAA و چاپ BBBB جابجا می‌شود. بعدا به لینوکس مبدل گشت!

اندیس

- تابع، تعریف تابع، شی تابع، سر، بدنه، پارامتر، فراخوانی تابع، آرگومان، متغیر محلی، مقدار بازگشتی، تابع برگشت‌دار، هیچ یا نان، پیمانه یا ماژول، گزاره ورود، شئ پیمانه، نمادگذاری نقطه، ترکیب، جریان اجراء، دیاگرام پشته، قاب، ردگیری traceback

تمرین

- شبه‌کد زیر تابع بازگشتی محاسبهٔ بم‌م دو عدد صحیح و مثبت و a>b>0 را نمایش می‌دهد. پیاده سازی کنید.

```
Alg. BMM(a, b):
    If b == 0:
            return a
    else:
            return BMM(b, a % b)
```

- الگوریتم بازگشتی محاسبهٔ جمع مربعات اعضای فهرستی را پیاده کنید.

منابع

A. Downey, Think Python2, How to Think Like a Computer Scientist, Green Tea Press, 2nd ed., 2015.

P. J. Deitel, H. Deitel, “Introduction to Python for Computer Science and Data Science: Learning to Program with AI, Big Data and the Cloud,” Pearson, 2021.

H-P Halvorsen, "Python Programming," 2019.