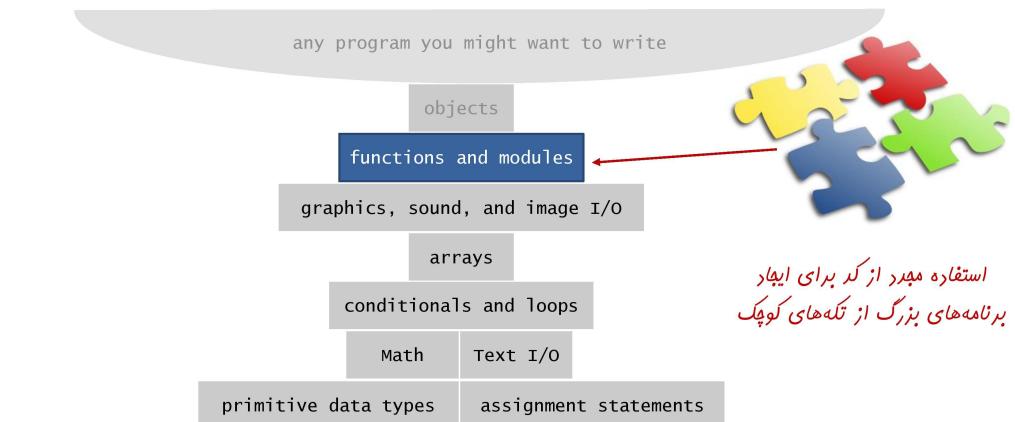


# عنصرهای برنامه‌نویسی

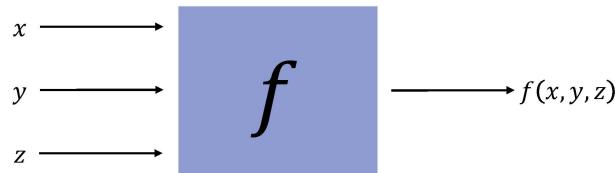


## تابع، کتابخانه‌ها و ماجول‌ها

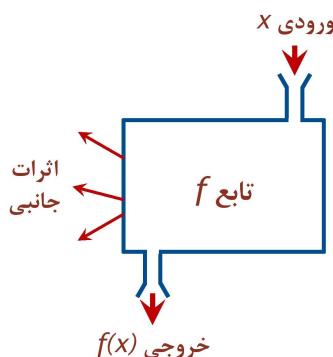
- ▣ برنامه‌نویسی ماجول‌لار (پیمانه‌ای).
- ▣ سازمان‌دهی برنامه به صورت مجموعه‌ای از ماجول‌های مستقل که در کنار یکدیگر کاری را انجام می‌دهند.
- ▣ چرا؟ سهولت اشتراک‌گذاری و استفاده مجدد از کد برای ایجاد برنامه‌های بزرگ.



## ۱-۱ توابع



## تابع



### تابع پایتون

- دریافت صفر یا چند کمیت به عنوان ورودی.
- برگرداندن صفر یا یک کمیت به عنوان خروجی.
- اثرات جانبی (مانند نوشتن در خروجی یا ترسیم نمودار).

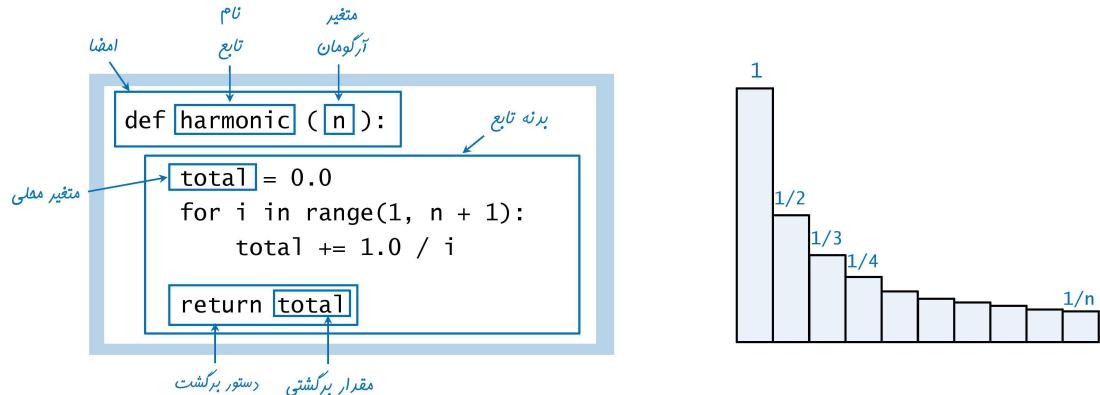
### کاربردها

- دانشمندان از توابع ریاضی برای محاسبه فرمول‌ها استفاده می‌کنند.
- برنامه‌نویس‌ها از توابع برای ایجاد برنامه‌های پیمانه‌ای استفاده می‌کنند.
- شما از تابع برای هر دو منظور استفاده می‌کنید.

### مثال‌هایی که تا کنون دیده‌اید

- توابع پیش‌ساخته: `math.sqrt()`, `random.random()`, `int()`, `str()`, `print()`
- کتابخانه‌های ورودی/خروجی: `stdaudio.play()`, `stddraw.line()`, `stdio.readInt()`

## ساختار یک تابع پایتون



## ساختار یک برنامه پایتون

```
import sys

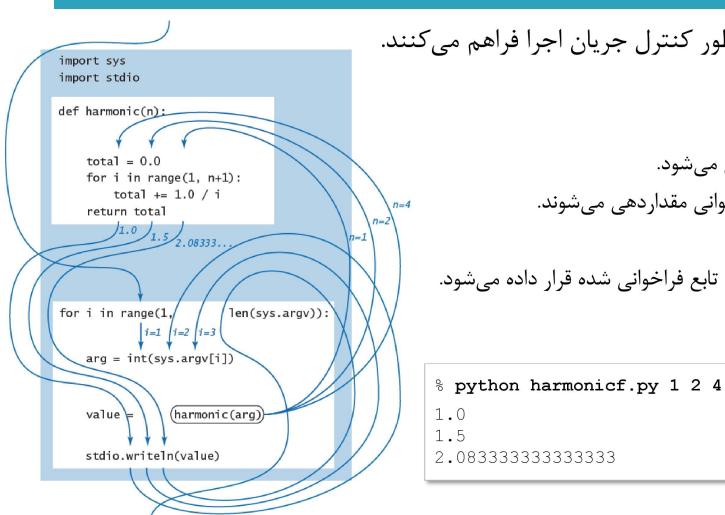
def harmonic(n):
    total = 0.0
    for i in range(1, n + 1):
        total += 1.0 / i
    return total

for i in range(1, len(sys.argv)):
    arg = int(sys.argv[i])
    value = harmonic(arg)
    print(value)
```

- ساختار یک برنامه پایتون.
- دنباله‌ای از دستورات `import`
- دنباله‌ای از تعریف توابع
- بخش سراسری یا بدن برنامه (اختیاری)

```
% python harmonicf.py 1 2 4
1.0
1.5
2.0833333333333333
```

## جريان کنترل



▪ نکته کلیدی. توابع یک روش جدید به منظور کنترل جریان اجرا فراهم می‌کنند.

### مراحل فراخوانی یک تابع.

- کنترل به ابتدای کد تابع فراخوانی شده منتقل می‌شود.
- آرگومان‌ها با استفاده از مقادیر موجود در فراخوانی مقداردهی می‌شوند.
- کد تابع اجرا می‌شود.
- به جای نام تابع در کد فراخوان مقدار برگشتی تابع فراخوانی شده قرار داده می‌شود.
- کنترل دوباره به کد فراخوان منتقل می‌شود.

## ایجاد توابع

▪ توابع به شما امکان می‌دهند یک سطح جدید از انتزاع ایجاد کنید:

- فراتر از آن چیزی که کتابخانه‌های از پیش بسته‌بندی شده در اختیار شما قرار می‌دهند.
- هر زمان بخواهید ابزار نیازتان را ایجاد می‌کنید: (`gaussian.Phi`) ...

▪ فرآیند.

- گام ۱: شناسایی یک ویژگی مفید
- گام ۲: پیاده‌سازی
- گام ۳: استفاده

▪ گام ۴: استفاده مجدد در هر برنامه‌ای که بخواهید.

In [ ]: █  
"""
function
"""

In [ ]: █ `def hello_1():
 print('hello world1')`  
`hello_1()`

```
In [ ]: ┌ def hello_2():
          return 'hello world2'

          s = hello_2()
          print(s)
```

```
In [ ]: ┌ def hello_3(p):
          print(p)

          s = 'hello world3'
          hello_3(s)
```

## بررسی نوع و پندریختی

□ چندریختی.

```
def f(a, b):
    return a + b
```

```
x = 2
y = 3
z = f(x, y) # z is 5
```

```
x = 'Hello, '
y = 'world'
z = f(x, y) # z is 'Hello, world'
```

```
In [ ]: ┌ def addtwo(a, b):
          return a+b

          print(addtwo(2, 3))           # 5
```

```
In [ ]: ┌ def f(a):
          a *= 2
          print(a)      #10
          return a+1

          b = 5
          r = f(b)
          print(r)      # 11
```

```
In [ ]: def f(x,y):
    if x > y :
        return x
    return y

r = f(2,5)
print(r) # 5
```

```
In [ ]: def g(x,y,z):
    return f(x , f(y,z))

print(g(2,5,3)) # 5
```

```
In [ ]: PI = 3.14

def area(r):
    return PI * r * r

def circumference(r):
    return 2 * PI * r

def main():
    r = 4
    print(area(r)) # 50.24
    print(circumference(r)) # 25.12

main()
```

## هزاره

هزاره (متغیر). مجموعه دستوراتی که می‌توانند به طور مستقیم به آن متغیر ارجاع دهند.

مثال: هزاره یک متغیر برابر است با کد پس از اعلان آن متغیر در درون بلوک کد.

```
def harmonic(n):
    total = 0.0
    for i in range(1, n + 1):
        total += 1.0 / i
    return total

for i in range(1, len(sys.argv)):
    arg = int(sys.argv[i])
    value = harmonic(arg)
    print(value)
```

این کد نمی‌تواند به ارجاع دهد  
هزاره *i*  
دو متغیر مختلف  
هزاره *i*  
هزاره *arg*, *value*,

```
In [ ]: ┌─ x = 1

def f():
    x = 2
    print(x)    # 2

f()
print(x)    # 1
```

```
In [ ]: ┌─ x = 1

def f():
    global x
    x = 2
    print(x)    # 2

f()
print(x)    # 2
```

```
In [ ]: ┌─ x = 5
def func():
    global x
    print(x)    # 5
    x = 8
    print(x)    # 8

func()
print(x)    # 8
```

```
In [ ]: ┌─ def f(a, b):
         a -=1
         b +=1
         return a, b

x = 3
y = 5
m, n = f(x, y)
print(m)    # 2
print(n)    # 6
```

# آرایه‌ها به عنوان آرگومان

<i>find the maximum of the arrays value</i>	<pre>def mean(a):     total = 0.0     for v in a:         total += v     return total / len(a)</pre>
<i>dot product</i>	<pre>def dot(a, b):     total = 0.0     for i in range(len(a)):         total += a[i] * b[i]     return total</pre>
<i>exchange two elements in the array</i>	<pre>def exchange(a, i, j):     temp = a[i]     a[i] = a[j]     a[j] = temp</pre>
<i>shuffle the array</i>	<pre>def shuffle(a):     n = len(a)     for i in range(n):         r = random.randrange(i, n)         exchange(a, i, r)</pre>

```
In [ ]: ┏ def f(a):
          a[0] -= 1
          a[1] += 1

          lst = [3, 5]
          f(lst)
          print(lst[0])    # 2
          print(lst[1])    # 6
```

```
In [ ]: ┏ def f(d):
          d['a'] -= 1
          d['b'] += 1

          my_dict = {'a':3, 'b':5}
          f(my_dict)
          print(my_dict['a'])    # 2
          print(my_dict['b'])    # 6
```

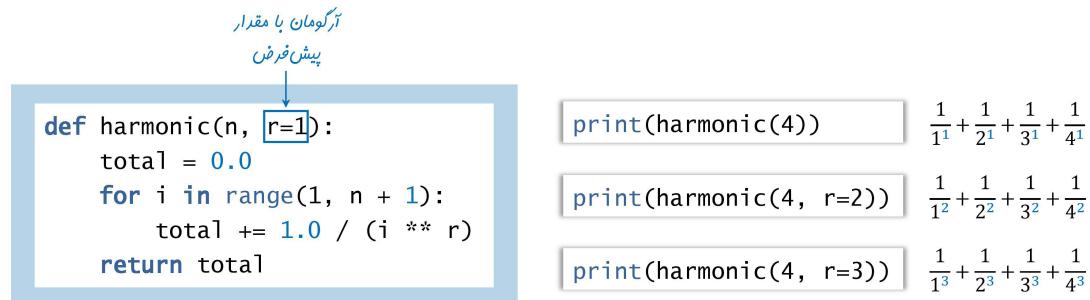
```
In [ ]: ┏ def f(a, b):
          print(a, b)

          f(1, 2)           # 1 2
          f(a = 1, b = 2) # 1 2
          f(1, b = 2)     # 1 2

          # f(2, a = 1)   # f() got multiple values for argument 'a'
```

# مقادیر پیشفرض برای آرگومان‌ها

□ آرگومان‌ها با مقادیر پیشفرض.



```
In [ ]: def f(a, b=5, c=7) :
          print(a,b,c)

f(1)           # 1  5  7
f(1, 3)         # 1  3  7
f(1, 3, 5)      # 1  3  5
f(1, c=9)       # 1  5  9
f(b=3, c=5 ,a=1) # 1 3  5

# f(1,b=2,5) # positional argument follows keyword argument
```

```
In [ ]: # keyword only argument

def f(*, a=3):
          print(a)

f()           # 3
f(a = 5)      # 5
#f(5)        # f() takes 0 positional arguments but 1 was given
```

```
In [ ]: ┌ # var arguments
```

```
def f(*t):
    s = 0
    for i in t:
        s += i
    print(s)

f(1, 2, 3)      # 6
f(5, 8)         # 13
```

```
In [ ]: ┌ def add(a, b, *more):
          r = a + b + sum(more)
          print(r)
```

```
add(1, 2, 3)    # 6
add(5, 8)        # 13
add(4, 5, 7, 9, 12) # 37
```

```
In [ ]: ┌ def f(a, b, *more):
          print(more)
```

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

```
In [ ]: ┌ def f(*t , x=9):
          print(x,t)

f(1, 2, x = 3)  # 3  (1, 2)
f(1, 2)          # 9  (1, 2)
f(1, 2, 5, 7)   # 9  (1, 2, 5, 7)
```

```
In [ ]: ┌ def concat(*s , sep='.'):
          return sep.join(s)
```

```
print(concat('ali', 'reza'))           # ali.reza
print(concat('ali', 'reza', 'sara'))    # ali.reza.sara
print(concat('ali', 'reza', 'sara', sep='/')) # ali/reza/sara
```

```
In [ ]: ┏ def f(a ,*, b=2, c=3):
      print(a,b,c)

f(1)          # 1  2  3
f(1, b=8)     # 1  8  3

# f(1, 3, c=4)
...
f() takes 1 positional argument but 2 positional arguments
(and 1 keyword-only argument) were given
...

# f(1, b=2, 4)    #  positional argument follows keyword argument
```

```
In [ ]: ┏ def f(a, b , **c):
      print(a,b,c)

f(3, 4, x=5, y=9)      # 3 4 {'x': 5, 'y': 9}
```

```
In [ ]: ┏ def f(a, b , *c , **d):
      print(a)  # 3
      print(b)  # 4
      print(c)  # (7, 1, 6)
      print(d)  # {'x': 5, 'y': 7, 'z': 9}

f(3 , 4, 7, 1, 6, x=5, y=7, z=9)
```

```
In [ ]: ┏ count_dict = {
      'L' : 0 ,
      'U' :0
  }

def func_count(s):
    for ch in s:
        if ch.islower():
            count_dict['L'] += 1
        else:
            count_dict['U'] += 1

s = 'FarSHid'
func_count(s)
print(count_dict['L'])    # 4
print(count_dict['U'])    # 3
```

```
In [ ]: ┏ def count_char(s):
      d = {}
      for i in s:
          if i in d.keys():
              d[i] +=1
          else:
              d[i] = 1
      return d

print(count_char('abbcfab'))      # {'a': 2, 'b': 3, 'c': 1, 'f': 1}
```

```
In [ ]: ┏ def count_word(s):
      d = {}
      words = s.split()
      for i in words:
          if i in d:
              d[i] += 1
          else:
              d[i] = 1
      return d

s = 'python created by rossum'
print(count_word(s)) # {'python': 1, 'created': 1, 'by': 1, 'rossum': 1}
```

```
In [ ]: ┏ ...
switch(a){
    case 1:return 'one' ;break;
    case 2:return 'two' ;break;
    defualt :return 'nothing';
}
...
```

```
In [ ]: ┏ def switch(a):
      d = {1:'one' , 2 : 'two'}
      return d.get(a,'nothing')

print(switch(1)) # one
print(switch(2)) # two
print(switch(8)) # nothing
```

```
In [ ]: ┌ grade_student = [
          {'id':1 , 'M':60 , 'F':40} ,
          {'id':2 , 'M':80 , 'F':70}
        ]

def ave_grade(lst):
    for d in lst:
        n1 = d.pop('M')
        n2 = d.pop('F')
        d['Ave'] = (n1 + n2) / 2
    return lst

print(ave_grade(grade_student))
# [{id:1 , Ave: 50.0} , {id:2 , Ave: 75.0}]
```

```
In [ ]: ┌ def reverse_string(s):
          r = ''.join(reversed(s))
          return r

print(reverse_string('abc'))      # cba
print(list(reversed('abc')))    # ['c', 'b', 'a']
```

```
In [ ]: ┌ def palindrome(s):
          return s == s[::-1]

print(palindrome('radar'))      # True
print(palindrome('ali'))        # False
```

```
In [ ]: ┌ def remove_index(s,start,end):
          if len(s) > end:
              s = s[0: start] + s[end+1 ::]
          return s

s= 'python'
print(remove_index(s,1,3))      # pon
```

```
In [ ]: ┌ def remove_oddindex(s):
          r = ''
          for i in range(len(s)):
              if i % 2 == 0:
                  r += s[i]
          return r

print(remove_oddindex('python'))    # pto
print(remove_oddindex('abcdefg'))  # aceg
```

```
In [ ]: ┌ def unique_list(lst):
    r = []
    for i in lst:
        if i not in r:
            r.append(i)
    return r

a = [1, 2, 3, 1, 4, 2]
print(unique_list(a))      # [1, 2, 3, 4]
```

```
In [ ]: ┌ def f(n):
    r = [1]
    for i in range(2, n):
        if (n % i) == 0:
            r.append(i)
    return r

lst = f(10)
print(lst)  # [1, 2, 5]
```

```
In [ ]: ┌ def f(s):
    w = ''
    lst = []
    for i in range(0, len(s)):
        if(s[i] != ' '):
            w += s[i]
        else:
            lst.append(w)
            w = ''
    m = lst[0]
    for j in range(0, len(lst)):
        if (len(lst[j]) > len(m)):
            m = lst[j]
    return m

s = 'python is an interpreted language.'
print(f(s))      # interpreted
```

```
In [ ]: ┌ ...
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
...
```

```
In [ ]: ┏ def pascal(n):
          t = [1]
          y = [0]
          for x in range(max(n,0)):
              print(t)
              t =[i+j for i,j in zip(t+y , y+t)]
      pascal(9)
```

```
In [ ]: ┏ def unique_list(lst):
          return list(set(lst))

          a = [1, 2, 3, 1, 4, 2]
          print(unique_list(a))           # [1, 2, 3, 4]
```

```
In [ ]: ┏ def f(s):
          my_set = set()
          w = s.split()
          for i in w:
              if i in my_set:
                  return i
              else:
                  my_set.add(i)
          return 'None'

          s = 'sara ali reza ali taha reza'
          print(f(s))    # ali
```

```
In [ ]: ┏ def prime(n):
          my_set = set()
          lst = []
          for i in range(2, n+1):
              if i in my_set:
                  continue
              for j in range(i*2, n+1, i):
                  my_set.add(j)
              lst.append(i)
          return lst

          n = 10
          print(prime(n))  #[2, 3, 5, 7]
```

```
In [ ]: ┌ def magic(m):
    n = len(m[0])
    lst = []

    lst.extend([sum(i) for i in m])

    for col in range(n):
        lst.append(sum(row[col] for row in m))

    r = 0
    for i in range(0,n):
        r += m[i][i]

    lst.append(r)

    r2 = 0
    for i in range(0,n):
        r2 += m[i][n-1-i]

    lst.append(r2)
    if len(set(lst)) > 1:
        return False
    return True

my_matrix = [
    [2, 7, 6],
    [9, 5, 1],
    [4, 3, 8]
]

print(magic(my_matrix)) # True
```

```
In [ ]: ┌ print('--- PEP 484 -----')

def greeting(name):
    return 'Hello ' + name

print(greeting('farshid')) # Hello farshid
```

```
In [ ]: ┌ def greeting(name: str) -> str:
    return 'Hello ' + name

print(greeting('farshid')) # Hello farshid
```

```
In [ ]: def add(x:int, y:int) ->int:  
    ''  
    sum two number  
    ''  
    print(x+y)  
  
add(2, 3)  
print(add.__annotations__)  
# {'x': <class 'int'>, 'y': <class 'int'>, 'return': <class 'int'>}  
  
print(add.__doc__) # sum two number
```

دانشگاه شهید مدنی آذربایجان  
برنامه نویسی مقدماتی با پایتون  
امین گلزاری اسکوئی  
۱۴۰۰-۱۴۰۱

[Codes and Projects \(click here\)](https://github.com/Amin-Golzari-Oskouei/Python-Programming-Course-Basic-2021) (<https://github.com/Amin-Golzari-Oskouei/Python-Programming-Course-Basic-2021>) [slides and videos \(click here\)](#) (<https://drive.google.com/drive/folders/1ZsQjBJJ4UAAp9zrGxm3c4qrhnvGBUYHw>).