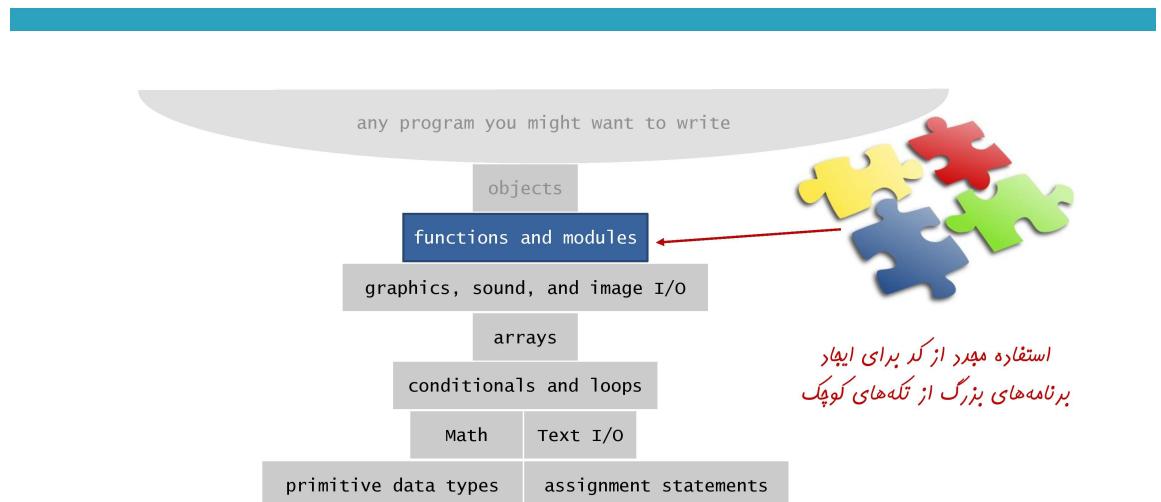


عناصر برنامه‌نویسی

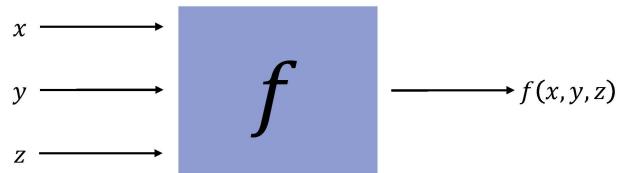


توابع، کتابخانه‌ها و ماجول‌ها

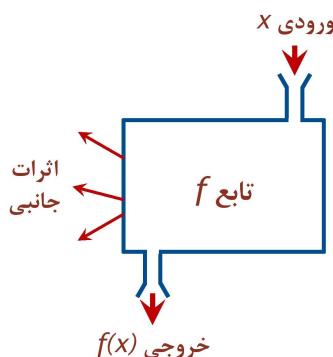
- ▣ برنامه‌نویسی ماجولار (پیمانه‌ای).
- ▣ سازمان‌دهی برنامه به صورت مجموعه‌ای از **ماجول‌های مستقل** که در کنار یکدیگر کاری را انجام می‌دهند.
- ▣ چرا؟ سهولت **اشتراک‌گذاری** و استفاده مجدد از کد برای ایجاد برنامه‌های بزرگ.



۱-۱ توابع



توابع



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).

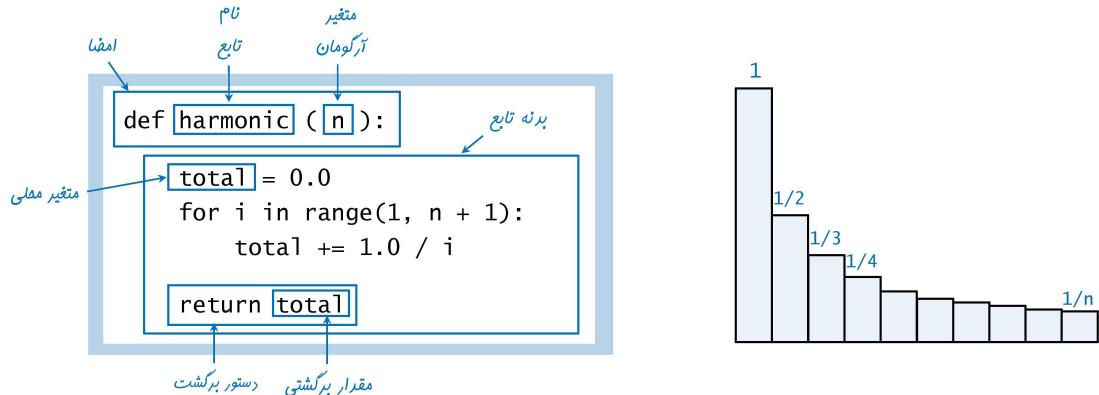
کاربردها.

- 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



```
In [ ]: ┏━━━━━━━━━━━━━━━
          print('function')
          ━━━━━━━━━━━━━━
```

```
In [ ]: ┏━━━━━━━━━━━━━
          def f():
                  print('Golzari')

          f()
          ━━━━━━━━━━━━
```

```
In [ ]: ┏━━━━━━━━━━━━━
          def g():
                  return 'Golzari'

          print(g())
          ━━━━━━━━━━━━
```

```
In [ ]: ┏━━━━━━━━━━━━━
          def h(p):
                  print(p)

          h('Golzari')
          ━━━━━━━━━━━━
```

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

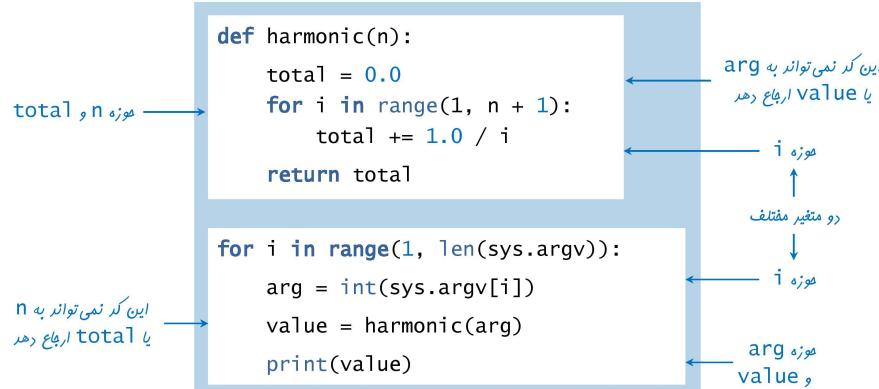
          def g(x,y,z):
                  return f(x , f(y,z))

          print(g(3,8,2))  # 8
          ━━━━━━━━━━━━
```

حوزه

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

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



```
In [ ]: x = 7
def func():
    global x
    print(x)    # 7
    x = 3
    print(x)    # 3

func()
print(x)      # 3
```

```
In [ ]: def f(a, b):
          a *= 2
          b += 3
          return a, b

x = 4
y = 6
m, n = f(x, y)
print(m)      # 8
print(n)      # 9
```

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(lst):
          lst[0] *= 2
          lst[1] /= 3

          a = [4, 9]
          f(a)
          print(a[0])    # 8
          print(a[1])    # 3.0
```

```
In [ ]: ┏ def h(my_dict):
          my_dict['x'] *= 2
          my_dict['y'] /= 3

          d = {'x':4 , 'y':9}
          h(d)
          print(d['x'])    # 8
          print(d['y'])    # 3.0
```

```
In [ ]: ┏ def test(x, y):
          print(x, y)

          test(5, 2)          # 5 2
          test(x = 5 , y = 2) # 5 2
          test(y = 2 , x = 5) # 5 2
          test(5 , y = 2)     # 5 2
```

مقادیر پیشفرض برای آرگومان‌ها

□ آرگومان‌ها با مقادیر پیشفرض.

```
آرگومان با مقدار
پیشفرض
def harmonic(n, r=1):
    total = 0.0
    for i in range(1, n + 1):
        total += 1.0 / (i ** r)
    return total
```

`print(harmonic(4))` $\frac{1}{1^1} + \frac{1}{2^1} + \frac{1}{3^1} + \frac{1}{4^1}$

`print(harmonic(4, r=2))` $\frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2}$

`print(harmonic(4, r=3))` $\frac{1}{1^3} + \frac{1}{2^3} + \frac{1}{3^3} + \frac{1}{4^3}$

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

print(harmonic(4))
print(harmonic(4, r=2))
print(harmonic(4, r=3))
```

```
In [ ]: # keyword only argument

def f(*, x=8):
    print(x)

f()          # 8
f(x = 2)    # 2
#f(3)       # f() takes 0 positional arguments but 1 was given
```

```
In [ ]: # var arguments

def add_more(a, b, *c):
    print(a + b + sum(c))

add_more(5, 2, 7, 8, 12)    # 5+2+7+8+12=
add_more(2, 6)              # 8
```

```
In [ ]: ┌ def f(*x , y='.'):
      ┌   return y.join(x)

      print(f('ali' , 'reza'))           # ali.reza
      print(f('ali' , 'reza' , 'sara' , y='/'))    # ali/reza/sara
```

```
In [ ]: ┌ def test(a ,*, b=7, c=9):
      ┌   print(a,b,c)

      test(5)          # 5 7 9
      test(1, b=4)     # 1 4 9

      # test(1, b=2, 6)    # positional argument follows keyword argument

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

```
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 [ ]: ┌ 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 [ ]: ┌ ...
      ┌ switch(a){
      ┌   case 1: return 'one' ;break;
      ┌   case 2: return 'two' ;break;
      ┌   default :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(3))  # nothing
```

```
In [ ]: ┏ a = [1, 2, 3, 1, 4, 2]

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

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

```
In [ ]: ┏ def unique_list_2(lst):
      return list(set(lst))

      print(unique_list_2(a))         # [1, 2, 3, 4]
```

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

      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)

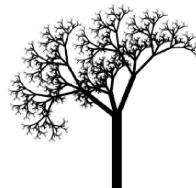
      add(2, 3)  # 5

      print(add.__annotations__)
      # {'x': <class 'int'>, 'y': <class 'int'>, 'return': <class 'int'>}

      print(add.__doc__)  # sum two number
```

توابع بازگشتی

- تابع بازگشتی. تابعی که خودش را به صورت مستقیم یا غیرمستقیم فراخوانی می‌کند.



- مزایای یادگیری توابع بازگشتی.

آشنایی با یک سبک جدید تفکر (تفکر بازگشتی)

آشنایی با یک الگوی قدرتمند برنامه‌نویسی



- رابطه نزدیک با استقرای ریاضی.

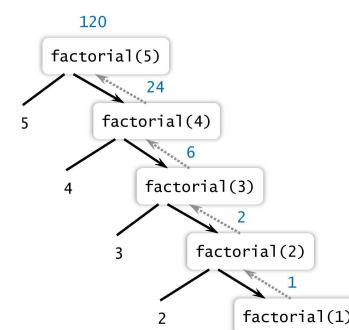
فاکتوریل

- محاسبه فاکتوریل n .

$$n! = 1 * 2 * \dots * (n - 1) * n$$

$$n! = \begin{cases} 1 & n = 1 \\ n \times (n - 1)! & n > 1 \end{cases}$$

```
def factorial(n):
    if n == 1: return 1 ← تعمیم بازگشت
    return n * factorial(n - 1) ← فراخوانی بازگشتی
```



```
In [ ]: █ #####  
         print('recursive')  
         #####
```

```
In [ ]: ❶ ...
iterative:
    n! = 1*2*3*...*n

recursive:
    n! = n * (n-1)!
    1! = 1

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

...
```

```
In [ ]: ❷ def fact(n):
    if n == 1:
        return 1
    else:
        return n * fact(n-1)

print(fact(4))      # 24
```

```
In [ ]: ❸ def f(n,base):
    s = '0123456789ABCDEF'
    if n < base:
        return s[n]
    else:
        return f(n//base , base) + s[ n % base]

print(f(25,16))      # 19

...
f(25,16) = f(1,16) + s[9] = 1 + 9 = 19
f(1,16)  = s[1]   = 1

...
print(f(8,2))        # 1000
```

بزرگ‌ترین مقسوم علیه مشترک

□ ب.م.م. یافتن بزرگ‌ترین عدد صحیحی که p و q بر آن بخش‌پذیرند.

```
gcd(4032, 1272) = 24.
```

$$\begin{aligned} 4032 &= 2^6 * 3^2 * 7^1 \\ 1272 &= 2^3 * 3^1 * 53^1 \\ \text{gcd} &= 2^3 * 3^1 \end{aligned}$$

□ مثال

□ کاربردها

□ ساده‌سازی اعداد کسری

□ RSA

□ رمزنگاری

```
In [ ]: def gcd(p,q):
    if q == 0:
        return p
    return gcd(q, p % q)

print(gcd(12,36))
```

```
In [ ]: def binary_search(lst, x, start=0, end=None):
    if end is None:
        end = len(lst) - 1
    if start > end:
        return False
    mid = (start + end) // 2
    if x == lst[mid]:
        return mid
    if x < lst[mid]:
        return binary_search(lst, x, start, mid - 1)
    return binary_search(lst, x, mid + 1, end)

a = [2, 4, 7, 12, 19, 25, 38]
print(binary_search(a, 19 ))      # 4
print(binary_search(a, 4 ))       # 1
print(binary_search(a, 20))       # False
```

توابع بازگشتی: فطاھای متداول (۱)

□ فراموش کردن حالت پایه (ختم بازگشت).

□ فراموش کردن حالت پایه، منجر به گیر افتادن در یک حلقه بی‌نهایت می‌شود.

```
def H(n):
    return H(n-1) + 1.0/n
```



□ عدم تضمین همگرایی.

□ استفاده از فراخوانی‌های بازگشتی برای حل زیرمسئلی که کوچک‌تر نیستند.

```
def H(n):
    if n == 1: return 1.0
    return H(n) + 1.0/n
```



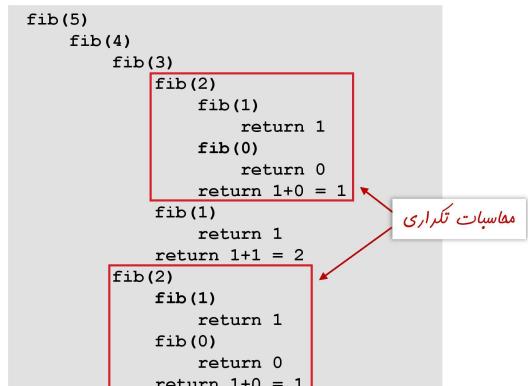
توابع بازگشتی: فطاھای متداول (۲)

□ محاسبات تکراری.

□ یک تابع بازگشتی ساده نیز ممکن است به دلیل محاسبات تکراری، دارای پیچیدگی زمانی نمایی باشد!

```
def fib(n):
    if n == 0: return 0
    if n == 1: return 1
    return fib(n-1) + fib(n-2)
```

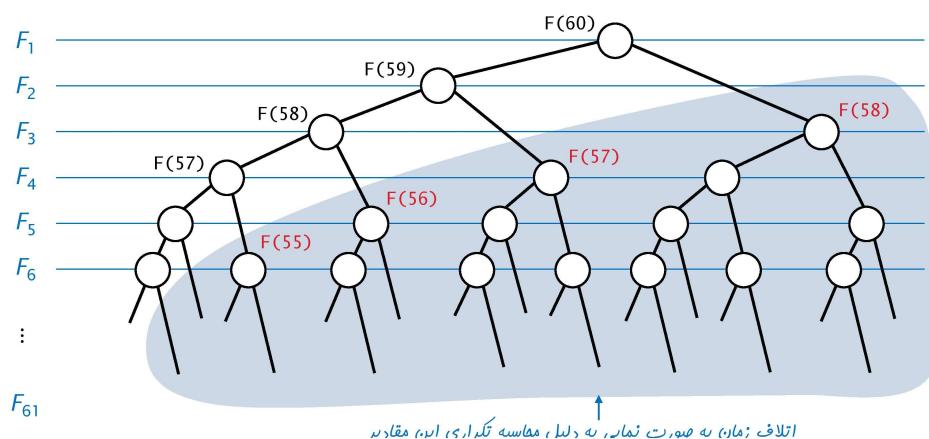
$$F_n = F_{n-1} + F_{n-2}$$



درخت فراخوانی بازگشتی برای اعداد فیبوناچی

فرض کنید C_n بیانگر تعداد دفعات فراخوانی $F(n)$ در هنگام محاسبه $F(60)$ باشد.

| n | C_n |
|-----|-----------------------|
| 60 | 1 |
| 59 | 1 |
| 58 | 2 |
| 57 | 3 |
| 56 | 5 |
| 55 | 8 |
| : | : |
| 0 | $>2.5 \times 10^{12}$ |



```
In [ ]: def fib(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    return fib(n-1) + fib(n-2)

print(fib(5))
```

اجتناب از اتلاف زمانی

```
import sys
memo = [0] * 200

def fib(n):
    if n == 0: return 0
    if n == 1: return 1
    if memo[n] == 0:
        memo[n] = fib(n-1) + fib(n-2)
    return memo[n]

n = int(sys.argv[1])
print(fib(n))
```

یادداشت برداری.

استفاده از یک آرایه به منظور به خاطر سپاری تمامی مقادیر محاسبه شده

اگر مقداری قبل محاسبه شده، آن را برگردان.

در غیر این صورت، آن مقدار را محاسبه کن، در آرایه ذخیره کن و سپس آن را برگردان.

```
% python fibonacciM.py 60
1548008755920
% python fibonacciM.py 80
23416728348467685
% python fibonacciM.py 100
354224848179261915075
```

```
In [ ]: memo = [0] * 200

def fib(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    if memo[n] == 0:
        memo[n] = fib(n-1) + fib(n-2)
    return memo[n]

print(fib(60))
```

برنامه‌ریزی پویا



□ برنامه‌ریزی پویا.

□ انجام محاسبات به شیوه «پایین به بالا»

□ ابتدا زیرمسائل کوچکتر را حل و راه حل آنها را ذخیره کن.

□ از این راه حل‌ها برای ایجاد راه حل‌های بزرگ‌تر استفاده کن.

```
def fib(n):
    F = [0] * (n + 1)
    F[0], F[1] = 0, 1
    for i in range(2, n + 1):
        F[i] = F[i - 1] + F[i - 2]
    return F[n]
```

| fib(5) | | | | | | |
|--------|---|---|---|---|---|--|
| 0 | 1 | 2 | 3 | 4 | 5 | |
| 0 | 1 | 1 | 2 | 3 | 5 | |

برنامه‌ریزی پویا



□ برنامه‌ریزی پویا.

□ انجام محاسبات به شیوه «پایین به بالا»

□ ابتدا زیرمسائل کوچکتر را حل و راه حل آنها را ذخیره کن.

□ از این راه حل‌ها برای ایجاد راه حل‌های بزرگ‌تر استفاده کن.

```
def fib(n):
    F = [0] * (n + 1)
    F[0], F[1] = 0, 1
    for i in range(2, n + 1):
        F[i] = F[i - 1] + F[i - 2]
    return F[n]
```

```
% python fibonacci.py 60
1548008755920
% python fibonacci.py 80
23416728348467685
% python fibonacci.py 100
354224848179261915075
```

```
In [ ]: ┏ def fib(n):
          F = [0] * (n + 1)
          F[0], F[1] = 0, 1
          for i in range(2, n + 1):
              F[i] = F[i - 1] + F[i - 2]

          return F[n]

print(fib(60))
```

```
In [ ]: ┏ #####lambda#####
print('lambda')
#####lambda#####
```

```
In [ ]: ┏ add = lambda x, y : x + y
print(add(2, 3)) # 6
```

```
In [ ]: ┏ f = lambda x, y : (x + y, x - y)
print(f(8, 1)) # (9, 7)
```

```
In [ ]: ┏ print('---map---')

lst = ['ALI', 'REZA']
print(list(map(str.lower, lst))) #['ali', 'reza']
```

```
In [ ]: ┏ a = ['mahsa', 'sara']
b = [20, 16]

print(list(zip(a,b))) # [('mahsa', 20), ('sara', 16)]

print(list(map(lambda x, y : (x,y) , a , b)))
```

```
In [ ]: ┏ a = [1, 3]
b = [2, 4]
print(list(map(lambda x, y :x+y, a, b))) # [3, 7]
```

```
In [ ]: ┏ a = [16, 7, 14, 6]
print(list(map(lambda x : x < 10 , a))) # [False, True, False, True]
```

```
In [ ]: ┏ print('--- filter ---')
a = [16, 7, 14, 6]
print(list(filter(lambda x : x < 10 , a))) # [7, 6]
```

```
In [ ]: ┏ a = [12, 17, '', 6, '' , 18]
print(list(filter(None,a))) # [12, 17, 6, 18]
```

```
In [ ]: ┏ print('--- reduce ---')

      └─ from functools import reduce

      lis = [12, 17, 6, 18]
      add = lambda a,b : a+b

      print(reduce(add, lis))      # 53   : (((12+17)+6)+18)
```

```
In [ ]: ┏ print('----- sorted -----')

      └─ lst = [12, 17, 6, 18]
          print(sorted(lst))      # [6, 12, 17, 18]
```

```
In [ ]: ┏ d = {'ali' : 12, 'sara' : 17, 'taha' : 6 , 'mahsa' : 18}
      └─ print(sorted(d.items() , key = lambda x : x[1]))

      # [('taha', 6), ('ali', 12), ('sara', 17), ('mahsa', 18)]
```

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

[Codes and Projects \(click here\)](https://github.com/Amin-Golzari-Oskouei/Python-Programming-Course-Advanced-2021) (<https://github.com/Amin-Golzari-Oskouei/Python-Programming-Course-Advanced-2021>)
[slides and videos \(click here\)](https://drive.google.com/drive/folders/1Dx3v7fD1QBWL-MNP2hd7ilxaRbeALkkA)
 (<https://drive.google.com/drive/folders/1Dx3v7fD1QBWL-MNP2hd7ilxaRbeALkkA>)