```
dicti = {'car': '021', 'year': 2000 , 'z': '36', 'p': 630}
dicti.sort()  ××false
sorted(dicti)  → ['car', 'p', 'year', 'z']
```

به متغیر سراسری فقط می‌توان از درون تابع دسترسی داشت؛ اما نمی‌توان آن را از درون تابع ویرایش کرد.

```
c = 0 # global variable

def add():

    global c
```

حال میتوان متغییر C را درون تابع ویرایش نیز کرد !!!

نکته مهم در ارسال ارگومان، توجه به چگونگی ان است!

در بین زبان‌های برنامه‌نویسی دو شیوه برای ارسال آرگومان رایج است: **"by value"** و **"by reference"**. در شیوه by value یک کپی از مقدار آرگومان به تابع ارسال می‌گردد و در نتیجه با تغییر مقدار پارامتر متناظر در تابع، مقدار آرگومان ارسال شده در خارج از تابع بدون تغییر باقی می‌ماند. به مثال پایتونی پایین توجه نمایید:

```
>>> def f(a):
...     a = 2
...     print(a*a)
...
>>> b = 3
>>> f(b)
4
>>> b
3
```

همانطور که در نمونه کد بالا قابل مشاهده است، مقدار متغییر b بدون تغییر باقی مانده است.

ولی در شیوه by reference به جای ارسال یک کپی از مقدار آرگومان، یک ارجاع (reference) به از آرگومان به تابع ارسال می‌گردد. می‌توان این‌طور در نظر گرفت که پارامتر متناظر در تابع، همان آرگومان در خارج از تابع است. در نتیجه با تغییر مقدار پارامتر متناظر در تابع، مقدار آرگومان در خارج از تابع نیز تغییر می‌کند. به مثال پایتونی پایین توجه نمایید:

```
>>> def f(a):
...     a[0] = 3
...     print(a)
...
>>> b = [1, 2]
>>> f(b)
[3, 2]
>>> b
[3, 2]
```

این دو از شیوه‌های مرسوم در زبان‌های برنامه‌نویسی هستند ولی ارسال پارامتر به صورت خاص در زبان برنامه‌نویسی پایتون چگونه است؟ در پایتون هر چیزی یک شی است و در نتیجه ارسال آرگومان‌ها در هر شرایطی به صورت **"by reference"** انجام می‌پذیرد.

و اگر سوال شود که علت تفاوت رفتار در دو مثال قبل چیست؟ باید بدانیم که علت به ماهیت اشیای آرگومان‌های ارسالی مربوط است. ارسال اشیای تغییرناپذیر (Immutable) به مانند انواع بولین، اعداد، رشته و تاپل به تابع، باعث بروز رفتاری مشابه با شیوه by value می‌شود ولی در مورد ارسال اشیای تغییرپذیر (Mutable) به مانند انواع لیست، دیکشنری و مجموعه اینگونه نخواهد بود. به

```
def outer():
    x = "local"

    def inner():
        nonlocal x
        x = "nonlocal"
        print("inner:", x)

    inner()
    print("outer:", x)


outer()
```

خروجی:

inner: nonlocal

outer: nonlocal

قواعد کلیدواژه global

- هنگامی که یک متغیر درون تابع ساخته می‌شود، به طور پیش‌فرض محلی است.
- هنگامی که یک متغیر بیرون از تابع تعریف می‌شود، به طور پیش‌فرض سراسری است و نیازی به استفاده از کلیدواژه global در پایتون نیست.
- از کلیدواژه global در پایتون برای خواندن و نوشتن یک متغیر سراسری درون یک تابع استفاده می‌شود.
- استفاده از کلیدواژه global بیرون از تابع، هیچ اثری ندارد.

تعریف متغیر بدون نوع: a = None

```
x='2.5'
int (x) → error
float (x) → ✓
---
```

```
list[2:5:2]  → [start:end:stride]
list[::2]
list.pop(<index-number>)
```

```
# Other functions
list.index(16)  # return 1
list.count(15)  # return 1, count the number of given value
```

```
list.insert(2, 30)  #list will change to: [15, 16, 30, 12, 14, 20, 11]
list.remove(15)  # remove 15 from index 0 of list
list.sort()  # sort the values in the list
list.extend(new_list)
```

```
set1.add('RF67')

# return new set contain unique value of both sets
Set1.union(new_codes)
# update set1
Set1.update(new_codes)

a.difference(b) ≈ a - b
```

```
data = tuple()
data.count(○) , data.index(○)
```

Dict

Keys can not be unhashable type such: 'list' or 'set' or …

.get vs .setdefault :

```
>>> users
{'name': 'Ali', 'weight': 60.0}
>>> users.get('age', 20)
20
>>> users
{'name': 'Ali', 'weight': 60.0}
>>> users.setdefault('age', 20)
20
>>> users
{'name': 'Ali', 'weight': 60.0, 'age': 20}
>>>
```

**** vs update :**

```
Dict1.update({3: "Scala"})
```

```
>>> users
{'name': 'Ali', 'weight': 60.0, 'age': 20}
>>> new_users
{'fullname': 'Hosein'}
>>> {**users, **new_users}
{'name': 'Ali', 'weight': 60.0, 'age': 20, 'fullname': 'Hosein'}
>>> users
{'name': 'Ali', 'weight': 60.0, 'age': 20}
>>> new_users
{'fullname': 'Hosein'}
```

```
del set ('name') vs set.pop('name')

Dict = {'Dict1': {1: 'Geeks'}, 'Dict2': {'Name': 'For'}}

print(Dict['Dict1'][1]) --> Geeks
print(Dict['Dict2']['Name']) --> For
```

---

printf

```
print (f" a is = {a}" , f" type a = { type (a)}") ≈
print (f" a is = {a}  type a = { type (a)}")
```

```
print('*' * 50)
```

50 تا ستاره بغل هم چاپ میکنه ↑

شرط یه خطی:

```
print("number id even") if number % 2 == 0 else print("Number is odd")
```

ساخت لیست یه خطی :

```
odd_list = [i for i in range(1, 50, 3) if i % 2 != 0]
```

خروجی دو خط زیر یکیه:

```
list4 = [i for i in my_list if i % 2 != 0 ]
list3 = list(filter(lambda i :   i % 2 != 0   , my_list ))
```

```
tmp = [j**2 for j in range(10)]
```

تابع

```
def greeting(name, age=20, *args, **kwargs)

# greeting('Ali', 18)  # Positional
# greeting(age=18, name="Ali")  # Keyword
# greeting("Ali", 21, 60, "Reza", {1, 2, 3}) --> optional
# greeting("Ali", age=21, wieght=60)
                          Args , kwargs فقط از داخل خود تابع قابل دسترسی هستند!
```

```
def announce(name, age, scores):
```

هر سه دستور زیر با هم یکیه !

```
# msg = announce(data['name'], data['age'], data['scores'])
# msg = announce(name=data['name'], age=data['age'], scores=data['scores'])

print(announce(**data))


data = [{"name": "Hosein", "age": 18, "scores": [12, 10, 14, 15]}]
```

اگه اون زرد ها با هم یکی نبود ارور میشدا تو نوع سوم (**data)

```
abs()
pow()
sum()
all()
any()
bin()
range()
reversed()
```

```
sorted()
round()
breakpoint()
```
c -> continue execution
q -> quit the debugger/execution
n -> step to next line within the same function
s -> step to next line in this function or a called function

```
complex()
dir()
divmod (13,3) → (4, 1)
```

enumerate()

```
list(enumerate(seasons, start=1))
[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]
```

```
eval('55+1') → 56
```

---

open()

Character	Meaning
'r'	open for reading (default)
'w'	open for writing, truncating the file first
'x'	open for exclusive creation, failing if the file already exists
'a'	open for writing, appending to the end of file if it exists
'b'	binary mode
't'	text mode (default)
'+'	open for updating (reading and writing)

```
msg = open('benevis.txt', 'a')
msg.write('salam azizam\n')
msg.close()
```

```
import csv
with open('c:\\Users\Mohammad Amin\documents\\0vsc\\grade.csv') as f:
    reader=csv.reader(f)
    with open ('d:\\averageeeeee.csv' , 'w' , newline='') as outfile :
        writer = csv.writer(outfile)
        for row in reader:
```

```
            name=row[0]
            new_sum, new_ave, count_of_grades=0,0,0

            for i in row[1:]:
                new_sum+=int(i)
                count_of_grades+=1
```

```
a, b ='11' ,'22'
print (a,b)
```

وقتی اینطوری پرینت کنیم در خروجی بین a و b یه فاصله میندازه :

```
11 22
```

```
win_msg = f"win: {team['result'].count('w')}".ljust(10)
```

تنظیم طول پرینت (به طول 10)

Lambda, map, filter

```
cars = [{'car': 'Ford', 'year': 2005},……]

def myFunc(dic):
  return dic['year']

car.sort(key = myFunc)
```

```
a=[ (5,10) , (20,1) , (4,3) , (15,14)]
a.sort(reverse=1 , key = lambda x : x[1])
sorted("This is a test string from Andrew".split(), key=str.lower)
```

ورودی key در حقیقت هر یک از عناصر لیسته

دقت شود myFunc یه ورودی میگیره ولی تو key = myFunc بهش ورودی ندادیم (خودش میفهمه ورودیش، هر یک از عناصر لیسته)

```
map(func, iter)
filter(function (1,0 as output) , sequence)


list1 = list( map( lambda x: x*2 , [1,2,3,4,5]  ) )
list2 = list( map( lambda x: 'big' if x>5 else 'small' , list1))
List3 = list(filter( lambda x:   x % 3 == 1  , list2))
```

فرمت sort

```
sorted( list ,  key= func , reverse = 1 or 0 )
```

```
from operator import itemgetter, attrgetter
```

اگه لیستی از دیک ها داشته باشیم : (age یکی از کلید های اون دیک عه)

```
print ("list sorting by age: " ,  sorted(lis, key=itemgetter('age')) )
print ("sorting by age and name: ", sorted(lis, key=itemgetter('age', 'name')))
```

یا مثلا اگه لیستی از تاپل ها داشته باشیم و بخوایم اول طبق کاراکتر شماره 1 و بعد شماره 2 سورتشون کنیم:

```
sor = sorted( student_tuples,  key=itemgetter(1,2))
```

itemgetter برای لیست و تاپل و ... که ایندکس داره استفاده میشه، attrgetter برای مشخص مردن اینکه کدام attribute از کلاس رو مد نظرمونه استفاده میشه.

```
sorted(student_objects, key=attrgetter('age'), reverse=True)
```

که student_objects یه instance از کلاس Student عه و age یه attribute در اون کلاس!

Try , except

```
try:
      assert  0 <= p <= 1600 ,'assertion error has been occurred'

except (ValueError):
      print("Invalid age")

except (NameError):
      print("Invalid variable")

except AssertionError as msg  :
      pass
except ZeroDivisionError :
      pass
except :
      pass

else:
    print("Well done!")
finally:
    print("Process finished")
    user_data.close()
```

Try , except اجباریه، else اختیاریه و فقط وقتی اجرا میشه که Try با موفقیت به پایان برسه، finally اختیاریه و چه Try با موفقیت به پایان برسه چه نرسه اجرا میشه.

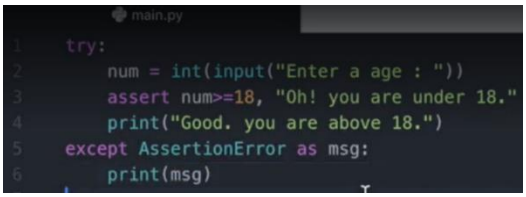
ValueError مثلا یه چیز نامربوط رو بخوایم به int تبدیل کنیم این اکسپشن raise میشه.

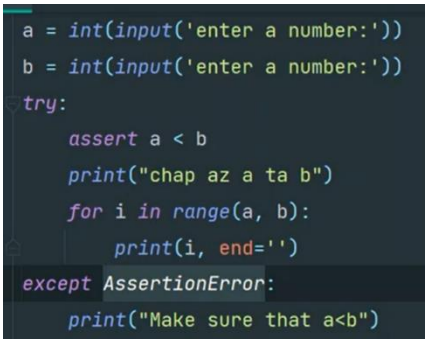
NameError مثلا یه متغییر رو تعریف نکرده باشیم و استفاده کرده باشیم.

---

raise ValueError()→

یه خطا خودمون از جنس ValueError تولید کردیم.





---

zip(*iterables)

ورودی هر چند تا iterables (یعنی چیزی که قابل پیمایش باشه) میتونه بگیره (اون لیسته که طولش کمتره، طول خروجی رو تعیین میکنه). خروجیش جنریتوره که هر عضوش تاپل تاپله.

`iterables`	**An iterable is anything you're able to iterate over (an iter-able). (like: list, string, dict), or user-defined iterables**

```
number_list = [1, 2, 3]
str_list = ['one', 'two', 'three']
result = zip(number_list, str_list)
print(set(result))

{(2, 'two'), (3, 'three'), (1, 'one')}
```

Unzip

```
c, v =  zip(*result)

c → (1, 2, 3)
```

```
v → ('one', 'two', 'three')
```