آموزش زبان برنامه نویسی پایتون

امیرحسین اسعدی

فهرست مطالب

پیش‌گفتار

خواننده گرامی کتاب پیش رو مقدمه ای است بر زبان برنامه نویسی پایتون نسخه ۳، و نیاز به پیش نیاز خاصی ندارد همراه با مثال های آموزشی که کمک شایانی به درک و فهم مفاهیم برنامه نویسی می کند. به نظر این جانب تسلط بر مفاهیم برنامه نویسی نیاز به تمرین عملی دارد توصیه می‌شود پس از مطالعه درس‌ها و مثال‌ها، آن ها را در سیستم پیاده کرده و اجرا کنید کد ها و یا حتی روند اجرای آن ها را مطابق میل خود تغییر داده ثابت ها را عوض کرده و خروجی را ببینید بازی کردن با کد ها به شما کمک می‌کند هرچه سریع تر به مفهوم مورد نظر مسلط شوید.
قطعا کتاب حاصل خالی از خطا نمی باشد خوشحال می شوم ایرادات و یا نظر های خود را و یا هر سخن دیگری با ایمیل زیر در ارتباط بگذارید.

a.asadi94@gmail.com

ارادتمند شما امیر اسعدی

فصل ۱

مقدمات

۱.۱ فلسفه پایتون

پایتونی زبانیست که به تازگی جایگاه ویژه‌ای در میان برنامه نویسان پیدا نموده است، یادگیری آسان، کاربران فعال و زیاد آن از طرفی کدهای از پیش آماده که به رایگان در اختیار همگان قرار دارد سبب شده است که این زبان رشد سریعی داشته باشد.
فلسفه پایتون بر اساس سادگی و کوتاه کردن کد های برنامه نویسی است.

۲.۱ نصب پایتون

محیط برنامه نویسی

برای برنامه نویسی کافیست تا از ویرایشگر های رایج مانند notepad++ استفاده کرده و سپس از طریق cmd برنامه خود را کامپایل کنید .

۱.۲.۱ ویندوز

ابتدا به سایت رسمی پایتون به آدرس www.python.org بروید . سپس از قسمت Downloads بر روی Windows کلیک کرده ، صفحه‌ای باز می‌شود که از ابتدا تا کنون تمامی نسخه های پایتون برای هر دو معماری ۳۲ و ۶۴ بیتی را دارد . آخرین نسخه را متناسب با معماری سیستم تان انتخاب نموده و پس از دانلود اقدام به نصب نمایید . در پنجره آغازین نصب تیک گزینه Add Python ۵.۳ to PATH را بزنید .

پس از اتمام مراحل نصب پنجره prompt command را باز کنید و python را تایپ کرده و سپس Enter را بزنید ، اگر با پیغامی مشابه به زیر رو برو شدید ، به این معناست که شما توانسته اید پایتون را نصب کنید .

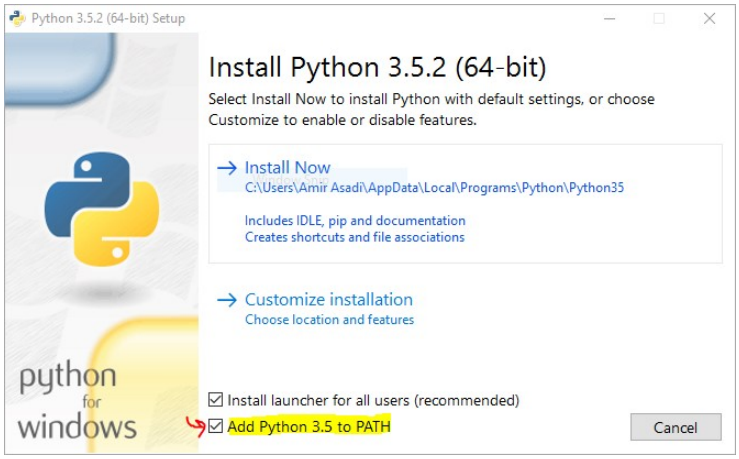
۲.۲.۱ گنو/لینوکس

اگر شما از کاربران گنو/لینوکس هستید باید تبریک بگویم زیرا به احتمال زیاد به صورت پیش‌فرض برای شما نصب شده است تنها کافیست تا وارد محیط ترمینال شوید و python را تایپ کنید تا ببیند نصب شده است یا خیر در صورت نیاز به نصب و یا بروزرسانی از دستور زیر استفاده نمایید

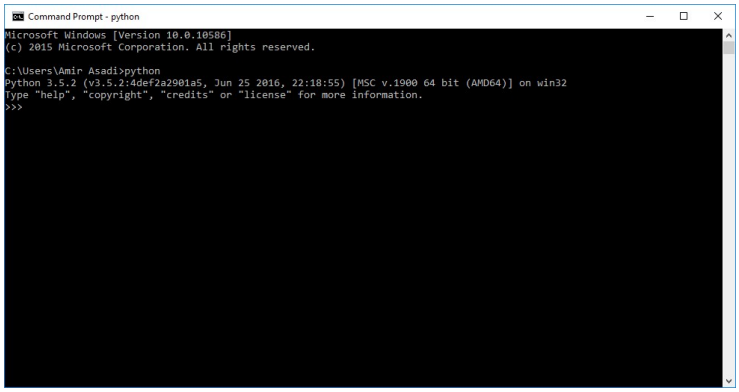
```
sudo apt-get install python
```

۳.۲.۱ مک اواس

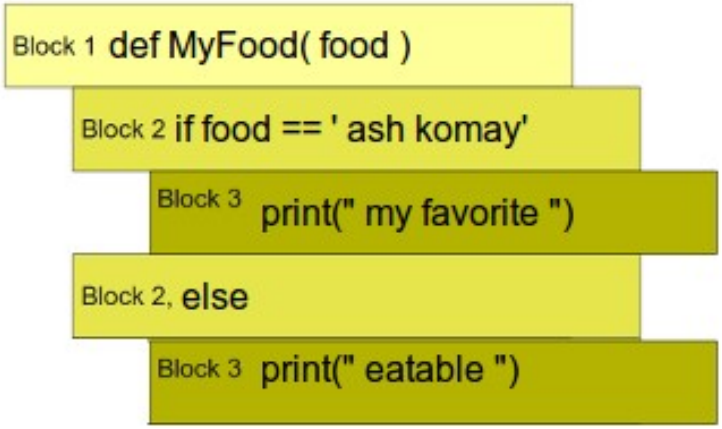
در سیستم عامل مک اواس نیز زبان پایتون به صورت پیش فرض نصب شده است،کافی است تا terminal را باز کنید و کلمه کلیدی python را وارد کنید سپس وارد محیط دوسویه برنامه نویسی می‌شوید.



شکل ۱.۱: محیط نصب در ویندوز



شکل ۲.۱: محیط پایتون در خط فرمان ویندوز



شکل ۳.۱: بلاک‌ها و فاصله گذاری در پایتون

۳.۱ فاصله گذاری ها

پایتون بلاک ها را با فاصله تشخیص می‌دهد و بر خلاف اکثر زبان‌های برنامه نویسی استفاده از و یا end begin دیگر لازم نیست. به مثال زیر دقت کنید

این کار سبب می‌شود تا کدهای نوشته شده الزاماً خوانا و زیباتر شوند . دو روش رایج برای فاصله گذاری ایجاد فاصله به اندازه ۴ space و یا استفاده از tab است .

فصل ۲

دستورات اصلی

۱.۲ کتابخانه ها

شما به راحتی می‌توانید کتابخانه‌های مورد نیاز خود را به برنامه اضافه کنید ، کافیست از دستور import و سپس اسم کتابخانه مورد نظر استفاده کنید . به مثال زیر دقت کنید :

```
import math
math.sqrt(81)

```

برای دیدن صحفه راهنمایی کتابخانه مورد نظر میتوان از دستور help به صورت زیر استفاده کرد help(math) و اگر تنها راهنمای تابع مرود نظرتان را می‌خواهید کافیست بعد از نام کتابخانه نقطه گذاشته و نام تابع را بیاورید help(math.factorial)

حال فرض کنید میخواهیم تعداد جایگشت ها ها را با توجه به فرمول زیر حساب کنیم

$\frac{n!}{k!(n-k)!}$

```
    import math
    n=5
    k=3
    math.factorial(n)/(math.factorial(k)*math.factorial(n-k))

```

همان طور که که می بینید به این روش شاید زمان زیادی بابت نوشتن جملات هدر رود راه دیگر مطابق روش زیر است

```
from math import factorial
n=5
k=3
factorial(n)/(factorial(k)*factorial(n-k))

```

و یا حتی خلاصه تر

```
from math import factorial as fac
n=5
k=3
fac(n)/(fac(k)*fac(n-k))

```

اگر میخواهید دو و یا بیشتر تابع خاص را وارد برنامه کنید آن ها را داخل پرانتز بنویسید و میانشان از ، استفاده کنید

```
from math import (factorial,sqrt)

```

و اگر میخواهید تمامی کتابخانه را وارد کنید به جای نام تابع از * استفاده کنید

```
from math import *

```

۲.۲ تقسیم

علگر / حاصل تقسیم دو عدد و یا متغییر را به صورت عدد اعشاری می دهد به عنوان مثال :

```
a=10
b=5
a/b
2.0

```

توجه شود عملگر // تنها قسمت صحیح تقسیم را می دهد به طور مثال :

```
a=11
b=5
a/b
2

```

۳.۲ تابع str

این تابع انواع مختلف را به نوع string تبدیل می کند .

۴.۲ تابع len

این تابع طول یک رشته را حساب می کند .

```
from math import factorial as fac
len(str(fac(100)))
158

```

۵.۲ انواع داده ها

۱.۵.۲ عدد صحیح int

اعداد صحیح با علامت را تا دقت بی نهایت ذخیره می کند دقت کنید که برای تعریف متغییر از نوع int نیازی به کلمه int نیست به مثال زیر دقت کنید a = ۲۰۲۰ برای تعیین عدد در مبنای ۲ از ۰b استفاده می کنیم برای تعیین عدد در مبنای ۸ از ۰o استفاده می کنیم برای تعیین عدد در مبنای ۱۶ از ۰x استفاده می کنیم تابع int انواع مختلف را به عدد صحیح تبدیل می کند برای تبدیل از مبنای ۱۰ به هر مبنای دیگری کافیست تا ورودی دوم تابع int مبنای مقصد باشد

```
0b10
2
0o10
8
0x10
16
int(-3.5)
-3
int("493")
493
int("110",2)
6

```

۲.۵.۲ عدد اعشاری float

۵۳ بیت برای ذخیره قسمت اعشار و ۱۵ تا ۱۶ بیت برای ذخیره قسمت صحیح عدد در نظر گرفته شده است - مطابق استاندارد IEEE-۷۵۴- .

- میتوان از نماد علمی برای نمایش اعداد خیلی بزرگ و یا کوچک استفاده کرده
- تابع float انواع مختلف را به عدد اعشاری تبدیل خواهد کرد
- حاصل عملیات بین دو عدد صحیح و اعشاری ، اعشاری خواهد شد

```
3e8
300000000.0
float(7)
7.0
float("3.14")
3.14
float("nan")
nan
float("-inf")
-inf
3.0 + 2
5.0

```

۳.۵.۲ هیچ None

نشان از نبودن داده را دارد و نماینگر هیچ و پوچ است توجه داشته باشید که در محیط تعاملی (REPL) چیزی نمایش داده نمی شود معادل null در زبان سی می باشد

```
None
a = None
a is None
True

```

۴.۵.۲ دودویی bool

این نوع داده برای ذخیره مقادیر منطقی به کار می رود که خواه می تواند درست و یا نادرست را ذخیره کند . تابع bool برای تبدیل انواع داده ای مختلف به نوع بول به کار می رود به مثال ها توجه کنید

- تنها عدد ۰ معادل False خواهد بود
- تنها رشته به طول ۰ معادل False خواهد بود
- تنها لیست خالی معادل False خواهد بود

```
bool(0.0)
False
bool(-1)
True
bool("False")
True
bool("")
False
bool([])
False
bool([1,5])
True
```

۶.۲ عملگرها

if a == b	برابری	==
if a != b	مخالف بودن	=!
if a < b	کمتر بودن	>
if a > b	بزرگتر بودن	<
if a >= b	بزرگتر مساوی بودن	=<
if a<= b	کوچکتر مساوی بودن	=>

۷.۲ عبارات شرطی

نحوه نوشتن آن به این صورت است

```
if condition :
  print("condition is True")
else
  print("condition if False")
```

در صورت درست بودن شرط وارد بلاک if شده و در غیر این صورت وارد بلاک else می شود ، توجه داشته باشید که عبارت داخل if و یا else نیاز به و ... ندارند بلکه با فاصله گرفتن - ۴ فضای خالی و یا یک tab – مشخص می شوند .

```
  h = 50
  if h>50 :
    print("Greater than 50")
  else
    print("50 or smaller")
  "50 or smaller"
```

۱.۷.۲ شرط‌های تو در تو

برای بررسی چند شرط می توانیم از یک if درون if دیگر استفاده کنیم به مثال زیر دقت کنید :

```
h = 50
if h>50 :
  print("Greater than 50")
else
  if h < 30
    print("less than 30")
  else
    print("between 30 and 50")
"between 30 and 50"

```

۲.۷.۲ دستور elif

در مثال بالا بهتر است else و if که زمینه زرد مشخص شده اند را ترکیب کرده و از دستور elif که مخفف (else if) می باشد استفاده کنیم .

```
h = 50
if h>50 :
  print("Greater than 50")
elif h < 30
  print("less than 30")
else
  print("between 30 and 50")
"between 30 and 50"

```

۸.۲ حلقه تکرار

حلقه های تکرار به منظور تکرار کردن یک دستور معین است به عنوان مثال شما میخواهید که کامپیوتر ده بار پیغام "hello world " را چاپ کند در این گونه موارد برای راحتی کار میتوانیم دستور for و یا while استفاده کنیم .

۱.۸.۲ حلقه while

دستور آن اینچنین است :

```
while condition :
  print("hello , world!")

```

تا زمانی که شرط condition درست باشد دستر داخل حلقه while تکرار خواهد شد ، توجه داشته باشید اینجا نیز نیازی به و .. نداریم و تنها با فاصله گزاری -۴ فضای خالی و یک tab - بلاک را مشخص می کنیم .

```
c = 5
while c != 0:
  print(c)
  c -= 1
5
4
3
2
1

```

۲.۸.۲ حلقه for

حلقه for مانند حلقه foreach در زبان سی شارپ است . ساختار for بدین صورت است :

```
for item in list
  body
```

به مثال زیر دقت کنید :

```
week = ['shanbe' , 'yekshanbe' , 'doshanbe' , 'seshanbe' , 'chharshanbe' , 'panjshanbe' , 'jome']
for day in week :
  print(day)

shanbe
yekshanbe
doshanbe
seshanbe
chharshanbe
panjshanbe
jome
```

در این مثال به جای لیست از دیکشنری استفاده کرده ایم :

```
shaeran = {'Saadi' : 7 , 'Khayyam' : 4 , 'Ferdosi' : 5 , 'Molavi' : 7 , 'Atar' : 6 , 'Hafez' : 8}
for shaer in shaeran :
  print('{0} dar gharne {1} miziste .'.format(shaer,shaeran[shaer]))

Molavi dar gharne 7 miziste .
Atar dar gharne 6 miziste .
Saadi dar gharne 7 miziste .
Ferdosi dar gharne 5 miziste .
Hafez dar gharne 8 miziste .
Khayyam dar gharne 4 miziste .
```

۳.۸.۲ حلقه بی‌نهایت

اگر می خواهید تا دستوری بی نهایت بار تکرار شود کافیست عبارت جلوی while همیشه True باشد مثل :

```
while True :
  print("infinite loop")
```

توجه داشته باشید در صورت برخورد با این گونه موارد برای توقف اجرای برنامه کلید های ctrl+c را فشار دهید .

۹.۲ دستور break

در صورت اجرا شدن این دستور ، اجرای مابقی دستورات بلاک رها شده و به بیرون از بلاک می رود

```
count = 0
while True :
  if count == 5 :
    break
```

```
        print(count)
        count +=1
    0
    1
    2
    3
    4

```

با رسیدن مقدار count به ۵ باتوجه به شرط داخل حلقه دستور break اجرا می شود و روند اجرای برنامه به بیرون از حلقه while خاتمه می شود .

۱۰.۲ گرفتن ورودی از کاربر با دستور input

تابع input() سبب می شود هر آن چه کاربر از طریق صفحه کلید وارد کند خوانده و در صورت نیاز ذخیره شود . به نحوه استفاده از این دستور دقت کنید :

```
x = input()
print(x)

```

هر آنچه کاربر وارد کند درون متغییر x ریخته می شود ، سپس با دستور print(x) متغییر x را چاپ می کند .

۱۱.۲ رشته ها String

رشته ها نوعی از داده اند که در خود متن ها و کارکتر ها را ذخیره می کنند ، برای تعریف یک رشته کافیست ابتدا اسمی دلخواه برای متغییرتان بگذارید ، سپس آن را برابر مقدار مورد نظر قرار دهید . رشته ها را در پایتون می توان مابین سینگل کوتیشن ' و یا دابل کوتیشن " قرار داد ، که این سبب می شود تولید رشته هایی مانند زیر آسان تر شود .

```
mystr = "It's a good thing."
print(mystr)
It's a good thing.
mystr = '"yes!", he said, "I agree!"'
print(mystr)
"yes!", he said, "I agree!"

```

توجه داشته باشید ایجاد رشته ای که میان دو علامت متفاوت قرار بگیرد ، باعث بروز خطا می شود .

mystr = "This is incorrect'

SyntaxError: EOL while scanning string literal

برای داشتن یک رشته با چند خط میتوان شروع و پایان متن را با سه علامت " و یا ' نشانه گذاری کرد .

```
mystr = """
This is a
multiline
string
"""

```

یا از \n استفاده کرد

```
mystr = "This is a \nmultiline\nstring"

```

۱.۱۱.۲ متد های string

متد capitalize

این متد ، اولین کاراکتر را با حرف بزرگ نشان می دهد

```
capital = 'tehran'
capital.capitalize()
'Tehran'

```

۲.۱۱.۲ : Sequence Escape

Sequence Escape ها رشته های خاصی هستند که در کنترل متن به ما کمک می کنند . مانند \n که در مثال قبل دیدیم . جدول نماد های مهم آن در زیر آمده است .

\\	Backslash (\)
\'	Single quote (')
\"	Double quote (")
\a	ASCII Bell (BEL)
\b	ASCII Backspace (BS)
\f	ASCII Formfeed (FF)
\n	ASCII Linefeed (LF)
\r	ASCII Carriage Return (CR)
\t	ASCII Horizontal Tab (TAB)
\v	ASCII Vertical Tab (VT)

۳.۱۱.۲ علامت r

این علامت باعث می شود رشته را همان گونه که هست بپذیرد . به عنوان مثال میخواهیم آدرس یک دایرکتوری را درون متغییر path بریزیم اگر از r استفاده نکنیم به این صورت خواهد بود

```
mystr = "C:\\Users\\Amir\\Downloads\\Compressed"

```

راه ساده تر آن اینگونه خواهد بود :

```
mystr = r"C:\Users\Amir\Downloads\Compressed"

```

شما می توانید به یک کارکتر خاص از رشته دسترسی داشته باشید ، در مثال زیر سومین کاراکتر را چاپ کرده ایم . (توجه داشته باشید معمولا شمارش در پایتون از صفر شروع می شود)

```
str = 'IRAN'
print(str[3])
N

```

توجه داشته باشید در زبان پایتون بر خلاف زبان C تفاوتی میان نوع داده ای ، یک کارکتر و آرایه از کارکتر ها وجود ندارد و تمامی آن ها string می باشند .

۱۲.۲ تابع type

با قرار دادن داده در ورودی تابع type می توانیم به نوع آن پی ببریم .

```
1       str = 'IRAN'
2       type(str[3])
3       <class 'str'>
4       type(str)
5       <class 'str'>
6
```

همان طور که می بینید خود str و چه یک کاراکتر از نوع str می باشند .

۱۳.۲ بایت‌ها byte

۱۴.۲ کامنت گذاری comment

فصل ۳

ساختمان های داده

۱.۳ لیست‌ها Lists

لیست ها محلی برای نگهداری اشیا مختلف هستند ، اجزا آن از طریق ، از هم جدا می شوند . لیستی از اعداد زوج کمتر از ده :

```
even = [0,2,4,6,8]
```

لیست چند میوه :

```
fruit = ['apple','orange','watermelon']
```

هم چنین همانند رشته ها میتوانیم از طریق اندیس گذاری به اجزای آن دسترسی داشته باشیم :

```
fruit[1]
'orange'
```

به راحتی می توانید عناصر لیست را با مقادیر دلخواه عوض کنید :

```
fruit[1] = 25
fruit
['apple', 25, 'watermelon']
```

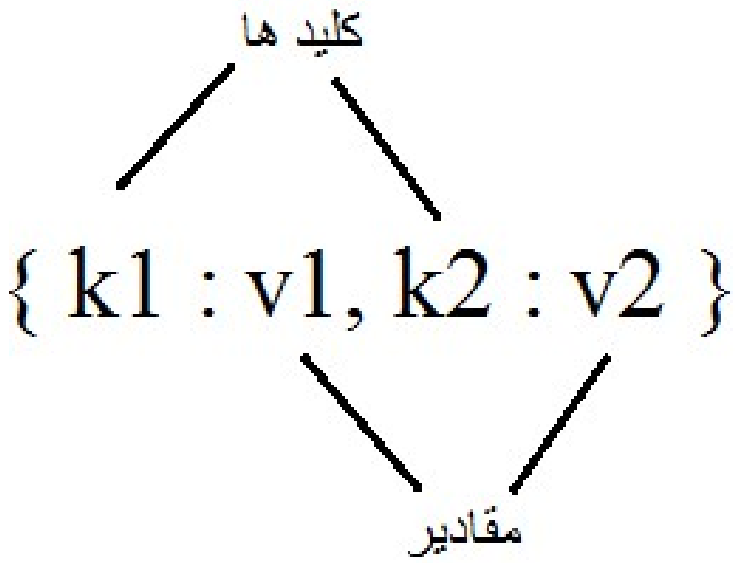
لیست خالی را به راحتی با مقدار نداد می توان ایجاد کرد .

```
city = []
```

۱.۱.۳ متد append

از این متد میتوان برای اضافه کردن عناصر به یک لیست استفاده کرد

```
city.append("Neyshabur")
city.append("Oslo")
city
['Neyshabur', 'Oslo']
```



شکل ۱.۳: نمایی از یک دیکشنری

۲.۱.۳ متد list

با استفاده از این متد می توانید انواع داده گوناگون را به لیست تبدیل نمایید :

```
list("Binalud")
['B', 'i', 'n', 'a', 'l', 'u', 'd']
```

۲.۳ دیکشنری Dictionary

دیکشنری ها کاربرد بسیار وسیعی دارند ، دیکشنری ها تعدادی کلید را به مقادیرشان نگاشت می کنند . نحوه ایجادشان در پایتون به این صورت است

```
tel = {'police':110,'atashneshani':125,'orzhans':911}
tel['police']
110
```

در صورت نیاز به تغییر مقدار یک کلید کافیست تا مقدار جدید را مطابق زیر وارد کنید

```
tel['orzhans'] =  115
tel
{'atashneshani': 125, 'police': 110, 'orzhans': 115}
```

اگر می خواهید کلید و مقدار جدیدی به دیکشنری خود اضافه کنید نام کلید و مقدار آن را مانند دستور زیر وارد کنید

```
tel['etelat']=113
tel
```

```
{'atashneshani': 125, 'etelat': 113, 'police': 110, 'orzhans':
911}
```

نحوه تعریف دیکشنری خالی مطابق زیر است

```
e = {}
```

۳.۳ توپل tuple

توپل ها دنباله ای immutable از اشیا هستند ، بدین معنا که هر گاه شی درون آن درست شود دیگر نمی شود آن را جایگزین و یا حذف نمود و یا عنصری جدید به این دنباله اضافه نمود . نحوه ایجاد توپل ها همانند لیست هاست ، با این تفاوت که به جای براکت از پرانتز استفاده می کنم .
ایجاد یک توپل

```
myTuple = ("NEVADA",365,0.01)
```

دسترسی به خانه اولین خانه آن :

```
myTuple[0]
'NEVADA'
```

در حقیقت پرانتز ها برای توپل های به اندازه یک و یا بیشتر اختیاری هستند

```
p = 1,2,0,1,1
type(p)
<class 'tuple'>
```

۱.۳.۳ تابع len

این تابع طول توپل را می دهد

```
len(myTuple)
3
```

چاپ اعضای یک توپل با حلقه for

```
for item in myTuple:
  print(item)

NEVADA
365
0.01
```

اضافه کردن به اعضای یک توپل

```
myTuple + (19.09,13e4)
('NEVADA', 365, 0.01, 19.09, 130000.0)
```

۲.۳.۳ عملگر *

شما با این عملگر می توانید اعضای تاپل را به تعداد دلخواه تکرار کنید

```
myTuple*3
('NEVADA', 365, 0.01, 'NEVADA', 365, 0.01, 'NEVADA', 365, 0.01)
```

توپل های چند بعدی :
به راحتی میتوانید با تعریف کردن یک توپل داخل توپل دیگر توپل های چند بعدی درست کنید ، در مثال زیر ما درون یک توپل ، ۱۲ توپل دیگر که نمایانگر ماه های سال و تعداد روز هایشان هستند درست کرده ایم :

```
month = ((1,31),(2,31),(3,31),(4,31),(5,31),(6,31),(7,30),(8,30)
,(9,30),(10,30),(11,30),(12,30))
```

به نحوه اندیس گذاری دقت کنید

```
month[7]
(8, 30)
month[7][1]
30
```

۳.۳.۳ توپل با یک عنصر

گاهی لازم است توپلی تنها با یک عنصر درست کنیم ، اما چگونه ؟ شاید بگویید کافیست مقدارمان را درون پرانتز بیاوریم . اما ببینید :

```
t = (255)
type(t)
<class 'int'>
```

راه حل آن آوردن ، درست بعد از مقدارمان است :

```
k = (255,)
type(k)
<class 'tuple'>
```

۴.۳.۳ توپل خالی

درون پرانتز را خالی بگذارید :

```
e = ()
type(e)
<class 'tuple'>
```

۵.۳.۳ کاربردهای توپل بدون پرانتز

بازگردانی چند مقدار از تابع : فرض کنید تابعی میخواهیم بنویسیم با نام minmax که کوچکترین و بزرگترین عضو یک لیست را برگرداند ، با این قابلیت شما به راحتی می توانید چندین مقدار را همزمان در پایتون برگردانید

```
def minmax(items):
  return min(items),max(items)

minmax([25,10,36,58,66,45])
(10, 66)
lower,upper=minmax([25,10,36,58,66,45])
lower
10
upper
66
```

۶.۳.۳ جابه‌جایی عناصر

فرض کنید میخواهیم مقدار دو عنصر را با یکدیگر با هم عوض کنیم ، در پایتون اینکار به زیبایی و سادگی انجام میشود همان طور که می بینید اینکار به سادگی انجام شده ، و نیازی به روش های دیگر همانند استفاده از متغییر سوم نمی باشد .

```
a='Ananas'
b='Bademjoon'
a,b = b,a
a
'Bademjoon'
b
'Ananas'
```

۷.۳.۳ متد tuple()

این متد برای تبدیل اشیا به tuple استفاده می شود

```
tuple([17,22,11,31])
(17, 22, 11, 31)
tuple('planetarium')
('p', 'l', 'a', 'n', 'e', 't', 'a', 'r', 'i', 'u', 'm')
```

۸.۳.۳ بررسی وجود و یا عدم وجود یک عنصر خاص در توپل

برای این که ببینم عنصر مورد نظرمان در توپل موجود هست یا نه کافیست in و یا in not استفاده کنیم

```
1372 in (1357,1380,1399,1372)
True
1372 not in (1357,1380,1399,1372)
False
```

۴.۳ رشته string

قبلا کمی در مورد رشته ها صحبت کرده بودیم اما اکنون در این بخش مفصل تر رشته ها را مرود بررسی قرار می دهیم . دنباله ای immutable از کارکتر ها (Unicode codepints)

۱.۴.۳ تابع len()

با این تابع می توانید طول یک رشته را بدست آورید

```
len('intergouvernementalisations')
27
```

الحاق دو یا چند رشته را میتوان با + انجام داد

```
'pass'+'port'
'passport'
```

۲.۴.۳ تابع join()

با استفاده از این تابع میتوانید رشته های خود را با مقداری مشخص با هم دیگر اتصال دهید

```
browsers = '-'.join(['firefox','safari','chrome','IE :)'])
browsers
'firefox-safari-chrome-IE :)'
```

۳.۴.۳ تابع split()

این تابع کاربردی به شما اجازه می دهد تا رشته مورد نظرتان را در کاراکتر های مدنظر از هم جدا کنید . به عنوان مثال رشته قبلی را میخواهیم در کارکتر - از هم جدا کنیم

```
browsers.split('-')
['firefox', 'safari', 'chrome', 'IE :)']
```

۴.۴.۳ تابع partition()

همین طور که از نام این تابع پیداست رشته را با رشته مورد نظر سه قسمت می کند در این مثال گفته ایم رشته sistanandbaluchistan را از قسمت and تقسیم کند

```
'sistanandbaluchistan'.partition('and')
('sistan', 'and', 'baluchistan')
```

و یا در این مثال رشته Tehran:Abadan را از : به سه قسمت تقسیم کرده و هر کدام را در متغییر مربوطه ریخته ایم

```
departure, separator, arrival = 'Tehran:Abadan'.partition(':')
departure
'Tehran'
arrival
'Abadan'
```

۵.۴.۳ متد format

ین تابع کاربرد های فراوانی از جمله کمک به چاپ در رشته ها دارد . متد format مقادیر دلخواه را درون رشته مورد نظر درج می کند . مکان رشته هایی که باید جایگزین شوند را با {} مشخص می کنیم

```
'the age of {0} is {1}'.format('Janati',89)
'the age of Janati is 89'
"the age of {0} is {1} . {0}'s birthday is on {2}".format('Janati',89,'February 22')
"the age of Janati is 89 . Janati's birthday is on February 22"
```

می توانید داخل {} را شماره گذاری نکنید ، حواستان باشد در این موارد به همان ترتیبی که ورودی داده اید چاپ می شوند

```
'Drink {} glasses of water a {}'.format(8,'day')
'Drink 8 glasses of water a day'
```

همچنین ممکن است آکولاد ها را نام گذاری کنید و سپس با توجه به نامشان آن ها را مقدار دهی کنید.
راه دیگر ایندکس گذاری است

```
'Coordinates of Tabriz {latitude} {longitude}'.format(latitude='38°04N',longitude='46°18E')
'Coordinates of Tabriz 38°04N 46°18E'
```

```
date = (1400,12,11)
'Today is {date[0]}/{date[1]}/{date[2]}'.format(date=date)
'Today is 1400/12/11'
```

میتوان از صفات ماژول ها هم به نحو زیر استفاده کرد

```
import math
'math constants : pi={m.pi}, e={m.e}'.format(m=math)
'math constants : pi=3.141592653589793, e=2.718281828459045'
```

۶.۴.۳ محدود کردن تعداد ارقام

در مثال قبل فرض کنید می خواهیم جهت نظم تنها تا سه رقم اعشار اعداد را نمایش دهیم کافیست از f#.: استفاده کنید که # همان تعداد ارقام است

```
'math constants : pi={m.pi:.3f}, e={m.e:.3f}'.format(m=math)
'math constants : pi=3.142, e=2.718'
```

۷.۴.۳ range()

دنباله ای از اعداد صحیح است . درست کردن با استفاده از فراخوانی سازنده آن یعنی range() صورت میگیرد

```
range(5)
range(0, 5)
```

توجه داشته باشید مقدار ورودی range() مقداریست که در آن توقف می کند

```
for i in range(5):
    print(i)

0
1
```

```
2
3
4
```

می توانیم مقدار شروع شمارش را هم به عنوان اولین ورودی به range بدهیم

```
range(5,10)
range(5, 10)
list(range(5,10))
[5, 6, 7, 8, 9]
```

گام‌های شمارش

```
list(range(0,10,2))
[0, 2, 4, 6, 8]
```

توجه داشته باشید برای چاپ یک عناصر یک لیست شاید از روش پایین استفاده کنید اما با توجه به امکاناتی که پایتون به شما داده است این روش خوبی نیست و بهتر است که مانند زیر حل شوند

```
mahsool = [2,3,3.1,2.9]
for i in range(len(mahsool)):
    print(mahsool[i])

2
3
3.1
2.9
```

```
for i in mahsool:
  print(i)

2
3
3.1
2.9
```

۸.۴.۳ enumerate()

اگر قصد شمارش دارید بهتر است از enumerate() استفاده کنید تا از range() . در حقیقت enumerate() توپلی به این صورت است (index , value)

```
t = [23,65,47,89,15]
for item in enumerate(t):
    print(item)

(0, 23)
(1, 65)
(2, 47)
(3, 89)
(4, 15)
```

حال آنکه با استفاده از خواص format و tuple توانایی مدیریت بیشتر هم دارید

```
for i,v in enumerate(t):
    print('i = {} , v={}'.format(i,v))

i = 0 , v=23
i = 1 , v=65
i = 2 , v=47
i = 3 , v=89
i = 4 , v=15
```

۵.۳ لیست‌ها list

مجموعه ای هستند برای نگهداری اشیا مختلف

```
s = 'Namayeshe nahve shomare gozari yek list'.split()
s
['Namayeshe', 'nahve', 'shomare', 'gozari', 'yek', 'list']
s[4]
'yek'
```

یکی دیگر از ویژگی های شماره گذاری ، شماره گذاری از انتهاست در این حالت داریم

-1	-2	-3	-4	-5	-6
Namayeshe	nahve	shomare	gozari	yek	list

```
s[-5]
'nahve'
```

۱.۵.۳ برش slicing

برای برش یک قسمت خاص از لیست و نمایش آن کافیست اندیس خانه شروع و اندیس خانه توقف را بدهیم

	start			stop	
-6	-5	-4	-3	-2	-1
0	1	2	3	4	5
Namayeshe	nahve	shomare	gozari	yek	list

```
s[1:4]
['nahve', 'shomare', 'gozari']
s[1:-2]
['nahve', 'shomare', 'gozari']
```

اما توجه داشته باشید خانه شروع و پایان انتخابی است و میتوانید آن را وارد نکنید در این صورت از ابتدا شروع می کند و یا تا انتها می رود

```
s[4:]
['yek', 'list']
s[:3]
['Namayeshe', 'nahve', 'shomare']
```

و یا انتخاب کل لیست

```
full_slice = s[:]
full_slice
['Namayeshe', 'nahve', 'shomare', 'gozari', 'yek', 'list']

```

۲.۵.۳ تکرار لیست

برای تکرار کافیست تعداد دفعات تکرار را در خود لیست به این صورت ضرب کنید

```
[0] * 9
[0, 0, 0, 0, 0, 0, 0, 0, 0]

```

۳.۵.۳ پیدا کردن عنصر خاص

برای جستجو در لیست از متد index استفاده می کنیم ، و در ورودی آن عنصر مورد نظر را وارد می کنیم در صورت موجود بودن اندیس آن را برای ما بر می گرداند

```
 w = 'The best preparation for tomorrow is doing your best today'.
split()
 w
 ['The', 'best', 'preparation', 'for', 'tomorrow', 'is', 'doing',
'your', 'best', 'today']
 i = w.index('preparation')
 i
 2

```

و اگر درون لیست موجود نباشد با این خطا مواجه خواهید شد

```
w.index('sdf')
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
  w.index('sdf')
ValueError: 'sdf' is not in list

```

۴.۵.۳ متد count()

با استفاده از این متد میتوانید تعداد دفعات تکرار یک عنصر در لیست را مشاهده کنید

```
w.count('best')
2

```

۵.۵.۳ بودن یا نبودن

برای بررسی اینکه آیا عنصری در لیست موجود هست یا نیست از اپراتور in و یا in not میتوانید استفاده کنید

```
3 in [0,2,4]
False
3 not in [0,2,4]
True

```

۶.۵.۳ حذف کردن از لیست

شما به روش های گوناگونی می توانید عنصر مورد نظر خود را لیست حذف کنید حذف با del اندیس عنصرتان را بدانید و با دستور del آن را حذف کنید در این مثال عنصر با اندیس ۵ یعنی 'itself' را حذف می کنیم

```
u = "Nothing is impossible, the word itself says 'I'm possible'!"
.split()
u
['Nothing', 'is', 'impossible,', 'the', 'word', 'itself', 'says',
"'I'm", "possible'!"]
del u[5]
u
['Nothing', 'is', 'impossible,', 'the', 'word', 'says', "'I'm", "
possible'!"]

```

۷.۵.۳ متد remove()

با این تابع نیز می توان یک شی را از لیست حذف نمود ، و دیگر نیازی به دانستن شماره اندیس آن نیست

```
u.remove('the')
u
['Nothing', 'is', 'impossible,', 'word', 'says', "'I'm", "
possible'!"]

```

و یا به کمک index

```
del u[u.index('word')]
u
['Nothing', 'is', 'impossible,', 'says', "'I'm", "possible'!"]

```

توجه داشته باشید که اگر عنصر مورد نظر شما در لیست نباشد ، در حالت remove با این خطا مواجه خواهید شد

```
u.reomve('cat')
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
  u.reomve('cat')
AttributeError: 'list' object has no attribute 'reomve'

```

۸.۵.۳ اضافه کردن به یک لیست با متد insert

کافیست مکان و شی مورد نظر را وارد کنید . در مثال زیر ما کلمه Believe را به ابتدای جمله -اندیس صفر- اضافه می کنیم

```
a = "you can and you're halfway there".split()
a
['you', 'can', 'and', "you're", 'halfway', 'there']
a.insert(0,'Believe')
a
['Believe', 'you', 'can', 'and', "you're", 'halfway', 'there']

```

حال میخواهیم متن را به صورت معمولی و در یک رشته ببینیم

```
' '.join(a)
"Believe you can and you're halfway there"

```

۹.۵.۳ الحاق دو لیست با یکدیگر

شما به راحتی با استفاده از اپراتور + می توانید دو لیست را با هم جمع کنید

```
a = [0,2,4]
e = [0,2,4]
o = [1,3,5]
s = e+o
s
[0, 2, 4, 1, 3, 5]
s += [6,7]
s
[0, 2, 4, 1, 3, 5, 6, 7]

```

۱۰.۵.۳ استفاده از متد extend

با متد extend نیز می توانید یک لیست را به لیست دیگر بچسبانید در مثال زیر اعداد ۸ و ۹ را به لیست قبلی اضافه کرده ایم

```
s.extend([8,9])
s
[0, 2, 4, 1, 3, 5, 6, 7, 8, 9]

```

۱۱.۵.۳ معکوس کردن یک لیست

شما می توانید یک لیست را با استفاده از متد reverse برعکس کنید

```
alphabet = ['a','b','c']
alphabet.reverse()
alphabet
['c', 'b', 'a']

```

۱۲.۵.۳ مرتب کردن لیست

برای مرتب کردن یک لیست به راحتی از متد sort استفاده کنید

```
numbers = [5,14,85,96,36]
numbers.sort()
numbers
[5, 14, 36, 85, 96]

```

و اگر میخواهید لیست را برعکس مرتب کنید کافیست مقدار ورودی reverse آن را True کنید

```
numbers.sort(reverse=True)
numbers
[96, 85, 36, 14, 5]

```

۶.۳ دیکشنری ها dictionary

تناظر یک به یک میان کلید های immutable به مقادیر mutable . نحوه کلی تعریف آن به نحو زیر است

```
      mydictionary = { k1:v1 , k2:v2 }
      mydictionary[k2]
      v2

```

توجه داشته باشید که کلید ها -در اینجا k۱ و -k۲ باید منحصر به فرد باشند در این مثال دیکشنری ای به نام urls درست کرده ایم و نام سایت را در قسمت کلید و آدرس آن مقدار همان کلید می باشد و برای دسترسی به آن دیگر خبری از شماره اندیس آن نیست بلکه کافی است نام کلید را بگوییم

```
      urls = {'google':'http://google.com',
          'youtube':'http://youtube.com',
          'facebook':'http://facebook.com'}
      urls['facebook']
'http://facebook.com'

```

راه دیگر ایجاد دیکشنری استفاده از سازنده dict() است

```
      names_and_ages = [('Aida',8) , ('Adham',22) , ('saman',28) , ('
    Janati',89)]
      d = dict(names_and_ages)
      d
      {'saman': 28, 'Janati': 89, 'Aida': 8, 'Adham': 22}

```

و یا به صورت مستقیم

```
      phonetic = dict(a='alfa' , b='beta')
      phonetic
      {'b': 'beta', 'a': 'alfa'}

```

در مثال زیر قصد داریم با حلقه for کلید ها و مقادیر شان را چاپ کنیم

```
      alphabet = dict(a='apple' , b='blue' , c='car')
      for key in alphabet:
          print('{key} => {value}'.format(key=key, value=alphabet[key])
    )

      c => car
      a => apple
      b => blue

```

همانگونه که می بینید ترتیبی در نمایش کلید ها وجود ندارد اگر میخواهید تنها مقادیر کلید ها را پیمایش کنید از متد values() استفاده کنید .

```
      for value in alphabet.values():
          print(value)

      car
      apple
      blue

```

و اگر تنها کلید ها مد نظرتان است از keys()

```
for key in alphabet.keys():
    print(key)

c
a
b
```

و اگر هم کلید و هم مقدارش به صورت جدا گانه از ()items استفاده کنید

```
for key,value in alphabet.items():
    print('{key} >> {value}'.format(key=key, value=value))

c >> car
a >> apple
b >> blue
```

۱.۶.۳ اپراتر های in و in not

از این اپراتر ها برای بررسی بودن و یا نبودن یک عنصر خاص می توانید در دیکشنری ها استفاده کنید

```
alphabet = dict(a='apple' , b='blue' , c='car')
'd' not in alphabet
True
'c' in alphabet
True
```

۲.۶.۳ حذف یک عنصر از دیکشنری

برای این کار از کلمه del استفاده می کنیم وسپس نام عنصر را میاوریم

```
del alphabet['c']
alphabet
{'a': 'apple', 'b': 'blue'}
```

۳.۶.۳ بررسی mutable و immutable بودن دیشکنری، کلیدها و مقادیرشان

کلید ها باید immutable ، مقادیرشون میتونن mutable باشن و البته که خود دیکشنری هم mutable هست .

```
elements = {'H':[1,2,3],
            'He':[3,4],
            'Li':[6,7],
            'Be':[7,9,10]}
elements['H'] += [4,5,6,7]
elements
{'He': [3, 4], 'H': [1, 2, 3, 4, 5, 6, 7], 'Li': [6, 7], 'Be':
[7, 9, 10]}
elements['B'] = [10,11]
elements
{'B': [10, 11], 'He': [3, 4], 'H': [1, 2, 3, 4, 5, 6, 7], 'Li':
[6, 7], 'Be': [7, 9, 10]}
```

همان طور که می بینید به ایزوتوپ های عنصر H افزودیم و به کل دیکشنری عنصر B را

۷.۳ ست‌ها sets

مجموعه ای از اشیا نامرتب منحصر به فرد و immutable ساختن set ها بسیار شبیه به دیکشنری ها است ولی با این تفاوت که داده ها تکی هستند

```
s = {5,55,555,5555}
s
{555, 5555, 5, 55}
type(s)
<class 'set'>
```

توجه داشته باشید تعریف ست خالی با استفاده از سازنده آن انجام می شود .

```
d = {}
type(d)
<class 'dict'>
e = set()
type(e)
<class 'set'>
```

توجه داشته باشید نحوه دیگر تعریف آن استفاده از سازنده ()set می باشد بدین صورت

```
s = set([1,3,5,7])
s
{1, 3, 5, 7}
```

ست ها همانند قوانین مجموعه ها در ریاضی ، اشیا تکراری را نادیده گرفته و حذف می کنند ، به مثال زیر توجه کنید

```
t = [0,2,2,4,4,4,4,8,8]
set(t)
{0, 8, 2, 4}
```

در زیر پیمایش یک ست را با حلقه for می بینید ، همانگونه که می بینید اشیا ترتیب خاصی ندارند بلکه به صورت دلخواه نمایش داده می شوند

```
for x in {11,7,5,3,2,1}:
  print(x)

1
2
3
5
7
11
```

بودن و نبودن در یک مجموعه را اپراتور های in و in not بررسی می کنیم

```
s = {2,5,3,1}
4 not in s
True
51 in s
False
```

اضافه کردن به سری ها با متد ()add انجام می شود توجه داشته باشید که عناصر تکراری نادیده گرفته می شوند

```
p = {5,10}
p.add(15)
p
{10, 5, 15}
p.add(20)
p
{10, 20, 5, 15}
p.add(5)
p
{10, 20, 5, 15}

```

اضافه کردین چندین متغییر به صورت یکجا با متد ()update صورت می گیرد

```
p.update([25,30,35])
p
{35, 5, 10, 15, 20, 25, 30}

```

حذف یک عنصر با متد ()remove انجام میشود ، کافیست درون پرانتز عنصر را ذکر کنید، حذف یک عنصر با متد ()remove انجام میشود ، کافیست درون پرانتز عنصر را ذکر کنید

```
p.remove(35)
p
{5, 10, 15, 20, 25, 30}
p.remove(34)
Traceback (most recent call last):
  File "<pyshell#41>", line 1, in <module>
  p.remove(34)
KeyError: 34

```

راه دیگر حذف کردن عناصر استفاده از متد ()discard است، توجه داشته باشید در این روش اگر عنصر مورد نظر -در این مثال ۲۹- وجود نداشته باشد به شما خطایی نمی دهد

```
p.discard(30)
p
{5, 10, 15, 20, 25}
p.discard(29)
p
{5, 10, 15, 20, 25}

```

۱.۷.۳ کاربرد ست ها در جبر

مجموعه های زیر را در نظر بگیرید

```
brown_eyes = {'Marjan','Yaser','Atefeh','Mahru','Kianoosh'}
black_hair = {'Yaser','Mahru','Kianoosh','Mina','Javad'}
have_dog = {'Yaser','Kianoosh'}
have_cat = {'Yaser','Atefeh','Kianoosh','Lale'}
o_blood = {'Mina','Javad','Atefeh','Marjan'}
b_blood = {'Kianoosh','Mahru'}
a_blood = {'Yaser'}
ab_blood = {'Javad','Lale'}

```

۲.۷.۳ متد union()

این متد در حیقیت اجتماع دو مجموعه را بر میگرداند یعنی تمامی اشیا که هم عضو مجموعه اول و دوم باشند

```
brown_eyes.union(black_hair)
{'Mahru', 'Yaser', 'Atefeh', 'Mina', 'Marjan', 'Kianoosh', 'Javad'}
brown_eyes.union(black_hair) == black_hair.union(brown_eyes)
True

```

۳.۷.۳ متد intersection()

```
brown_eyes.intersection(black_hair)
{'Yaser', 'Mahru', 'Kianoosh'}
brown_eyes.intersection(black_hair) == black_hair.intersection(brown_eyes)
True

```

این متد اشتراک دو مجموعه را بر می گرداند یعنی اشیا مشترک ما بین دو مجموعه

۴.۷.۳ متد difference()

این متد تفاضل دو مجموعه را نشان می دهد

```
black_hair.difference(brown_eyes)
{'Mina', 'Javad'}
black_hair.difference(brown_eyes) == brown_eyes.difference(black_hair)
False

```

۵.۷.۳ متد symmetric_difference()

```
black_hair.symmetric_difference(brown_eyes)
{'Atefeh', 'Mina', 'Marjan', 'Javad'}
black_hair.symmetric_difference(brown_eyes) == brown_eyes.symmetric_difference(black_hair)
True

```

۶.۷.۳ متد issubset()

این متد بررسی می کند که آیا یک مجموعه ، زیرمجموعه دیگری هست یا خیر ؟

```
have_dog.issubset(black_hair)
True

```

۷.۷.۳ متد issuperset()

```
have_cat.issuperset(have_dog)
True

```

۸.۷.۳ متد isdisjoint()

این متد بررسی می کند آیا دو مجموعه کاملا از هم مجزا هستند ؟ به عنوان مثال یک فرد نمی تواند همزمان دو گروه خونی داشته باشد

```
1       a_blood.isdisjoint(o_blood)
2       True
3
```

فصل ۴

مباحث پیشرفته

۱.۴ خروج از محیط REPL

در ویندوز از Z + Ctrl و در لینوکس و مک از D + Ctrl استفاده کنید .

۲.۴ فایل‌ها در پایتون

برنامه های پایتون در فایلی با پسوند py ذخیره می شوند

۳.۴ تعریف تابع

تابع ها در حقیقت اجزای تشکیل دهنده برنامه ما هستند و وظایفی که به آن ها محول شده را با هر بار فراخوانی انجام می دهند . فرض کنید میخواهیم برنامه ای بنویسیم که تعداد جایگشت ها را حساب کند .

$$\frac{n!}{k!(n-k)!}$$

اگر مجاز به استفاده از تابع های پیش فرض پایتون نباشیم ، نیاز است تا سه بار فاکتوریل را بدست آوریم و این سبب می شود کد های برنامه ما زیاد شود . اما راه دیگر استفاده از تابع است کافیست یک بار برنامه فاکتوریل را نوشته و هر بار آن را صدا بزنیم . در زیر دستور تعریف کردن تابع و سپس فراخوانی آن در پایتون آمده است :

```
1       def function_name (arguments):
2         body
3
4       function_name(arguments)
5
```

همان گونه که می بینید ابتدا کلمه کلیدی def و سپس نام تابع که هرچیزی که دوست دارید میتواند باشد ، و درون پرانتز ورودی های آن را می دهیم و در نهایت دونقطه را قرار می دهیم . و در فراخوانی آن تنها نام تابع را می آوریم و درصورت نیاز ورودی های آن را می دهیم تابعی که سلام می کند

```
1       def func1():
2         print('Hello , world!')
3
4       func1()
5       Hello , world!
6
```

تابعی که توان دوم را حساب می کند

```
def square(x):
  return x*x

print(square(3))
9
```

اگر نیاز داشتید که مقداری را برگردانید از دستور return استفاده کنید تابعی که زوج و یا فرد بودن را تشخیص می دهد

```
def even_or_odd(n):
if n%2 == 0:
  print("Even")
else:
  print("Odd")

even_or_odd(10)
Even
even_or_odd(11)
Odd
```

۴.۴ نوشتن help

۱.۴.۴ نوشتن help برای کتابخانه

متن مورد نظرتان را در اول برنامه ما بین دو """ قرار دهید

۲.۴.۴ نوشتن help برای توابع

کافیست تا بعد از تعریف تابع متن مورد نظر را میان دو """ قرار دهید مانند :

```
def square(x):
"""
calculate x to the power of 2
Args:
  x an input number
Returns:
  x times x
"""
return x*x

help(square)
Help on function square in module __main__:

square(x)
  calculate x to the power of 2
  Args:
    x an input number
  Returns:
    x times x
```

۳.۴.۴ کامنت گذاری comment

کامانت ها توضیحاتی هستند در مورد کد هایی که نوشته شده ، و نحوه عملکرد برنامه را توضیح می دهند ، و هیچ تاثیری در روند اجرای برنامه نخواهند داشت . در پایتون کامنت را با علامت # شروع می کنند و تا پایان جمله ادامه می دهند

```
#this is comment

```

استفاده از shebang

بهتر است اولین خط برنامه تان این دستور باشد

```
#!/usr/bin/env/python

```

این سبب می شود که اجرا کننده برنامه (مفسر) در سیستم عامل یونیکس و یا لینوکس بفهمد که باید از مفسر پایتون برای اجرای برنامه استفاده کند . از این رو برای اجرای برنامه می توانیم در ترمینال به راحتی بنویسیم

```
bash\$ ./foo.py

```

این در حالی است که اگر از آن دستور استفاده نکنیم باید اینگونه برنامه را اجرا کنیم

```
bash\$ python ./foo.py

```

۵.۴ ماژول چیست module

کد های پایتونی که درون یک فایل py. قرار گرفته باشند ماژول نامیده می شود . ماژول های می توانند مستقیما اجرا شوند مانند

python module_ name.py

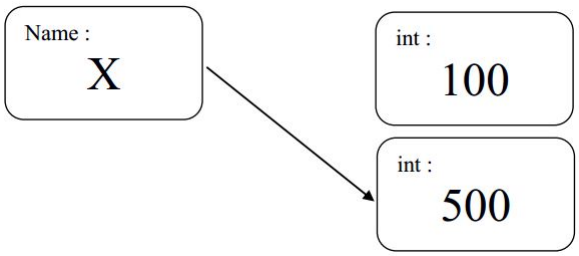
و یا در محیط REPL و یا درون یک ماژول دیگر با دستور import و سپس نام ماژول استفاده شوند اما حواستان باشد پسوند py. را به کار نبرید

Import module_ name

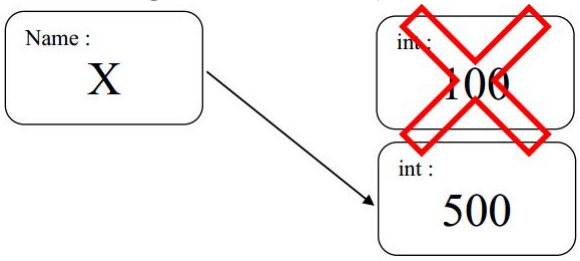
۶.۴ object to reference

ما در گذشته راجع به متغییر ها در پایتون صحبت کردیم ، اما به راستی متغییر ها چه هستند ؟ زمانی که شما یک عدد صحیح را درون یک متغییر می ریزید چه اتفاقی درون کامپیوتر می افتد ؟ فرض کنیم دستور x = ۱۰۰ را اجرا کنیم . پایتون یک شی از نوع int با مقدار ۱۰۰ درست می کند و یک شی مرجع دیگر با نام x

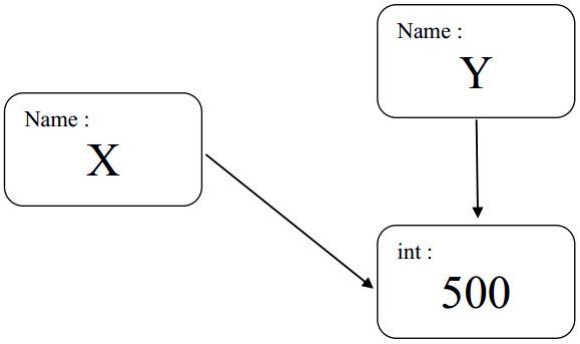
در حقیقت مقدار دهی سبب می شود یک اسم (در اینجا (x ، به یک object (دراینجا عدد ۱۰۰(مربوط شود. حال اگر دستور x = ۵۰۰ را اجرا کنیم ، با توجه به اینکه اشیا int در پایتون immutable هستند مقدار ۱۰۰ به ۵۰۰ عوض نخواهد شد ، در حقیقت اتفاقی که می افتد پایتون یک شی immutable ، int دیگری با مقدار ۵۰۰ درست می کند ، و اکنون شی x به ۵۰۰ اشاره می کند



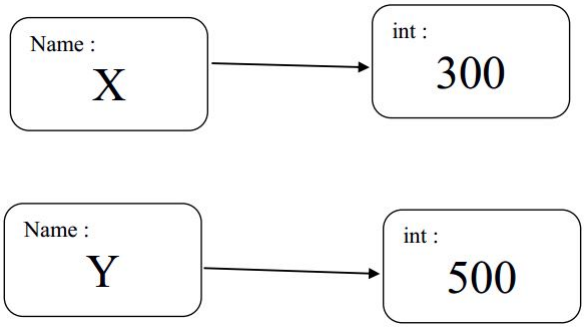
حال collector garbage پایتون به صورت اتوماتیک شی int با مقدار ۱۰۰ را حذف می کند



در دستور x = y ، شی رفرنس دیگری با نام y درست می شود و به ۵۰۰ اشاره می کند .



حال x = ۳۰۰ را اجرا می کنیم ،



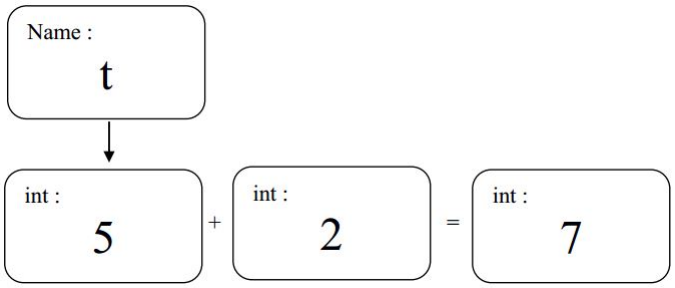
۷.۴ متد ID

این متد یک عدد صحیح و منحصر به فرد در طول دوره عمر متغییر مورد نظر را بر میگرداند

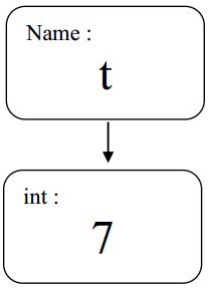
```
a = 111
id(a)
1850740624
b = 4
id(b)
1850737200
b = a
id(b)
1850740624
id(a) == id(b)
True
a is b
True
```

Listing 4.1: خطوط سه و شش متفاوتند حال آنکه سه و نه برابرند

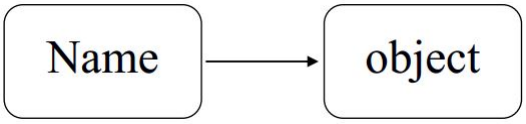
```
t = 5
id(t)
1850737232
t += 2
id(t)
1850737296
```



و در پایان collector garbage ۲ و ۵ را حذف می کند

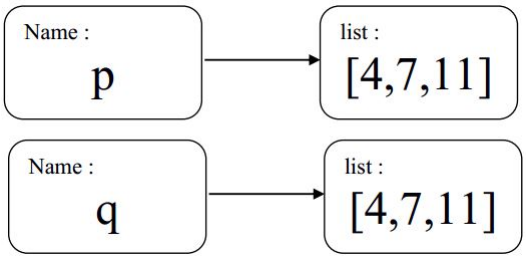


این رفتار پایتون برای تمامی انواع داده به کار می رود . متد id به شی ای که ایجاد شده است بر میگردد نه به مرجع آن در حقیقت اسم ها مانند برچسب هایی هستند که با استفاده از آن ها ، به اشیا دسترسی پیدا می کنیم



برای تفهیم بیشتر به این مثال دقت کنید : دقت کنید که دو شی مقادیرشان یکسان است

```
p = [4,7,11]
q = [4,7,11]
id(p)
3032666205576
id(q)
3032665493704
p == q
True
p is q
False
```



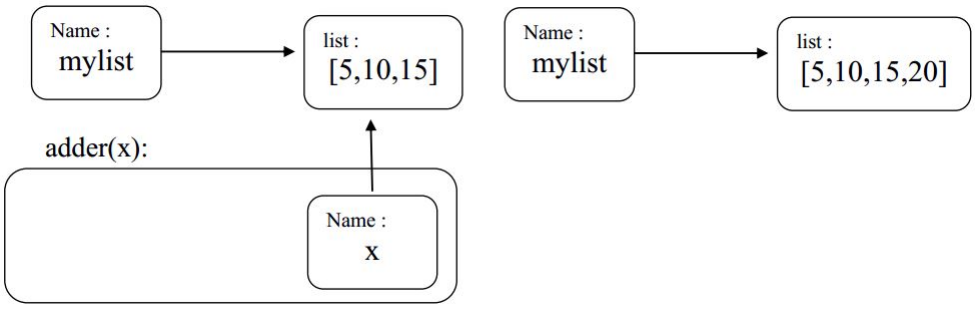
در حقیقت برابری محتوی دو متغییر سبب برابری مقادیر می شود ، اما دو شی که id آن ها یکی است براسی اشیا آن ها با هم یکیست .

۸.۴ ارسال آرگومان به تابع

مثال زیر را در نظر بگیرید

```
mylist = [5,10,15]
def adder(x):
  x.append(20)
  print("x = ",x)


adder(mylist)
x =  [5, 10, 15, 20]
mylist
[5, 10, 15, 20]
```

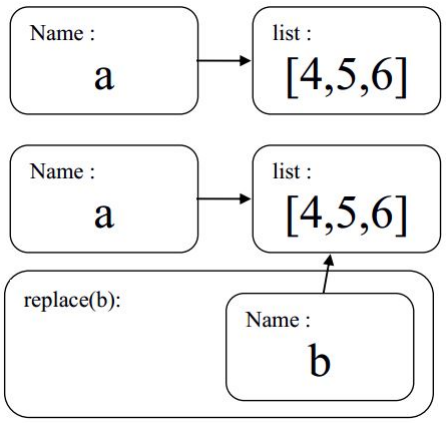
هنگامی که شی به تابعی ارسال می شود ، یک رفرنس دیگر از او ساخته می شود که دقیقا همانند او رفتار می کند id) هایشان یکسان است) ، در اینجا تابعی با نام adder داریم که هر لیستی که بگیرد ۲۰ را به آن اضافه می کند در ادامه تابع را فراخوانی می کنیم و سپس مقدار mylist را بررسی می کنیم همان طور که می بینید عدد ۲۰ به آن اضافه شده است



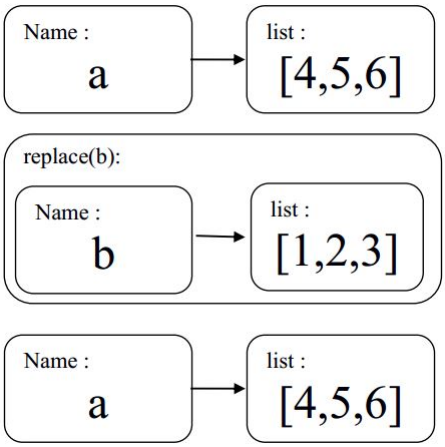
حال به این مثال جالب توجه کنید :

```
a = [4,5,6]
def replace(b):
  b = [1,2,3]
  print(" b = ",b)


replace(a)
 b =  [1, 2, 3]
a
[4, 5, 6]

```

همانند بالا تابعی مینویسیم که یک لیست از ورودی بگیرد ، ولی این دفعه بجای اضافه کردن به آن لیستی با مقادیر جدید به آن نسبت می دهیم



خط اول ابتدا a با مقادیر ۴و۵و۶ پر شده ،سپس متغییر جدید b ایجاد شده که ابتدا همان ۴و۵و۶ است و این دو شی دقیقا یکی هستند

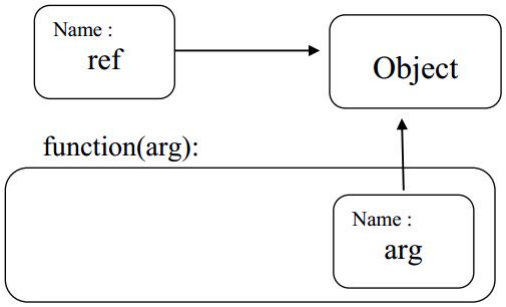


پس از مقدار دهی با مقادیر جدید ۱و۲و۳ دیگر به مقادیر قبلی اشاره نمی کند و وقتی چاپ می کنیم قطعاً مقادیر گم شده قبلی را چاپ نمی کند بلکه ۱و۲و۳ را چاپ می کند ، و وقتی از تابع بیرون می آییم مقادیر قبلی توسط garbage collector پاک شده و a همچنان ۴و۵و۶

اما حال اگر بخواهیم مقادیر یک لیست را تماما عوض کنیم راه حل چیست ؟ همان مثال قبلی را کمی توسعه می دهیم تا این بار هرگاه لیستی را ارسال کنیم مقادیر آن واقعا عوض شده و تاثیر آن در بیرون از تابع هم باقی بماند

```
a = [4,5,6]
def replace_contents(b):
  b[0]=1
  b[1]=2
  b[2]=3
  print("b = ",b)

a
[4, 5, 6]
replace_contents(a)
b =  [1, 2, 3]
a
[1, 2, 3]

```

ارسال آرگومان ها به توابع در پایتون به روشی به نام ، pass by object reference صورت می گیرد ، بدین معنا که مقدار منبع کپی می شود نه مقدار شی



۱.۸.۴ دادن مقدار پیش فرض به آرگومان های تابع

شما میتوانید به آرگومان های ورودی توابع خود مقادیر پیش فرض بدهید تابع function ۲ آرگومان ورودی دارد a و b ، که b مقدار پیش فرض value۱ را گرفته . به چند روش مختلف میتوانید تابعتان را فراخوانی کنید فقط مقادیر اجباری را بدهید مقدار پیش فرض را عوض کنید ترتیب مقادیر پیش فرض را عوض کنید به این مثال توجه کنید :

```
def function(a, b=value1)
    body

function(a)
function(a,b=value2)
function(b=value2,a)

```

همان گونه که می بینید توانسته ایم به چند روش مختلف تابع chap را فراخوانی کنیم و آرگومان هایش را مقدار دهی کنیم

```
def chap(message , border='-'):
  line = border * len(message)
  print(line)
  print(message)
  print(line)

chap("parchame se rang")
----------------
parchame se rang
----------------
chap("Work hard , Dream big","*")
**********************
Work hard , Dream big
**********************
chap("Work hard , Dream big",border="*")
**********************
Work hard , Dream big
**********************
chap(border=".",message="Blue dot")
........
Blue dot
........

```

آرگومان های ورودی تنها یک بار محاسبه و اجرا می شوند

فرض کنید میخواهیم تابع show_time را بنویسیم که هر بار زمان جاری را چاپ کند و به آن آرگومان ورودی پیش فرض می دهیم برای اینکار از متد ctime در ماژول time استفاده می کنیم

```
  import time
  def show_time(arg=time.ctime()):
    print(arg)

  show_time()
  Wed Aug 31 12:23:29 2016
  show_time()
  Wed Aug 31 12:23:29 2016
  show_time()
  Wed Aug 31 12:23:29 2016

```

پس از فرخوانی تابع برای اولین بار همه چیز درست است و زمان به درستی چاپ می شود ولی در دفعات بعدی با اینکه زمان سیستم گذشته است ولی هر باز زمان اول را چاپ می کند . پس نتیجه می گیریم ، مقادیر پیش فرض آرگومان های تابع تنها یک بار محاسبه می شوند و آن زمانیست که تابع محاسبه می شود

در این مثال تابعی نوشتیم به نام add_ spam() که آرگومان پیش فرض ورودی آن لیست خالی است ، با هر بار فراخوانی آن spam جدیدی به لیست اضافه می شود و اثر آن باقی می ماند .

```
def add_spam(menu=[]):
  menu.append("spam")
  return menu

sobhane = ["omlet","halim"]
add_spam(sobhane)
['omlet', 'halim', 'spam']
add_spam()
['spam']
add_spam()
['spam', 'spam']
add_spam()
['spam', 'spam', 'spam']

```

برای رفع این مشکل این گونه اقدام می کنیم

```
def add_spam(menu=None):
  if menu is None:
    menu = []
  menu.append('spam')
  return menu

add_spam()
['spam']
add_spam()
['spam']
add_spam()
['spam']

```

کافیست مقدار پیش فرض آرگومان ورودی را None بدهیم از این پس با فراخوانی تابع add_spam دیگر اثرات فراخوانی های قبلی در آن مشاهده نمی شود

۹.۴ نمایش زمان با ماژول time

برای گرفتن زمان جاری سیستم ابتدا ماژول time را به برنامه خود وارد کنید سپس از متد ctime استفاده کنید

```
import time
time.ctime()
'Wed Aug 31 12:13:59 2016'

```

۱۰.۴ scope ها در پایتون

وقتی یک متغییری ایجاد می کنیم ، نام آن در کجا ذخیره می شود ؟ در scope مربوط به خودش . ما چهار scope متفاوت در زبان پایتون داریم :

- Local : درون تابع فعلی

- Enclosing : تمامی تابع های پیوستی
- Global : در بالاترین سطح دسترسی در ماژول
- Built-in : توسط ماژول built-in در دسترس است

۱۱.۴ متغییر های global

وقتی تابع set_count فراخوانی می شود و مقدار ۵ به آن پاس داده می شود میخواهد آن را درون متغییر count بریزد . توجه داشته باشید count داخل تابع با count بیرون از تابع متفاوت است و یکی نیستند .

```
count = 0
def set_count(c):
  count = c

count
0
set_count(5)
count
0
```

اگر بخواهیم به count بیرون از تابع دسترسی داشته باشیم کافیست از global استفاده کنیم

```
count = 0
def set_count(c):
  global count
  count = c

count
0
set_count(5)
count
5
```

تفاوت این مثال با قبلی این است که دیگر پایتون درون تابع set_count متغییر جدیدی ایجاد نمی کند و از همان متغییر global count استفاده می کند و مقدار آن را برابر c قرار می دهد همان طور که می بینید پس از فراخوانی آن مقدار count از صفر به پنج تغییر مقدار داده است .

۱۲.۴ تعیین نوع متغییر

برای تعیین نوع متغییر از تابع type() استفاده می کنیم کافیست نام متغییر را به تابع بدهیم

```
mystr = "bonjour!"
myint = 98
def myfunc():
  pass

type(myint)
<class 'int'>
type(mystr)
<class 'str'>
type(myfuncfunc)
<class 'function'>
```

۱۳.۴ تابع dir

ین تابع تمامی صفت ها ، متدها ، متغییر ها و ... یک شی را به نمایش می گذارد

```
dir(mystr)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__'
, '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__',
'__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '
__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '
__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '
__subclasshook__', 'capitalize', 'casefold', 'center', 'count', '
encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit', '
isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', '
istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans
', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition'
, 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip',
 'swapcase', 'title', 'translate', 'upper', 'zfill']

```

همان طور کی می بینید تمامی ابزار هایی که یک متغییر از نوع str دارد را به نمایش گذاشته

۱۴.۴ مدیریت خطاها و استثنا

گاهی به دلیل خطای برنامه نویسی روند اجرای برنامه متوقف خواهد شد ، از این رو از استثنا ها در پایتون استفاده می کنیم به عنوان مثال تابعی می نویسیم با نام convert که وظیفه تبدیل به نوع عدد صحیح 'int' را دارد ، اگر به آن مقدار '۶۶ و یا هر عدد دیگری را بدهیم برنامه درست کار خواهد کرد اما با دادن ورودی رشته ای مانند 'susangerd' روند اجرای برنامه با خطا مواجه خواهد شد

```
def convert(s):
  x = int(s)
  return x

convert('66')
66
convert('susangerd')
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
  convert('susangerd')
  File "<pyshell#3>", line 2, in convert
  x = int(s)
ValueError: invalid literal for int() with base 10: 'susangerd'

```

اما چگونه این خطاها را مدیریت کنیم ؟ تابع convert را کمی تغییر می دهیم به این صورت

```
def convert(s):
  try:
    x = int(s)
  except ValueError:
    x = -1
  return x

```

بلاک try محلی است که در آن امکان رخ دادن خطا وجود دارد و بلاک except مکانی است که در هنگام بروز خطا اجرا می شود
با نوشتن پیغام برنامه را کامل تر می کنیم

```
def convert(s):
  try:
    x = int(s)
    print("Conversion succeeded! x =", x)
  except ValueError:
    print("Conversion failed!")
    x = -1
  return x

```

حال برنامه را با مقادیر مختلف امتحان می کنیم

```
convert('365')
Conversion succeeded! x = 365
365
convert('sparrow')
Conversion failed!
-1

```

توجه داشته باشید با دادن ورودی 'sparrow' دیگر دستور print پایین آن اجرا نشده و به بلاک except منتقل شده است