فصل اول : مقدمه

ویژگیهای مهم زبان برنامه نویسی پایتون :

* یک زبان همه منظوره

. طراحی سایت های داینامیک

. تولید نرم افزارهای دستکتاپ

. طراحی رابط کاربری

. هوش مصنوعی

. بازی سازی

. شبکه و امنیت

. داده کاوی و ساخت برنامه های محاسباتی

* یک زبان سطح بالا
* یک زبان با امکان یادگیری سریع
* یک زبان اسکریپتی ( مفسری )
* یک زبان شی گرا
* یک زبان رایگان و متن باز
* یک زبان توسعه پذیر و جای پذیر

. استفاده از کد زبانهای دیگر

. استفاده از کد زبان پایتون در زبانهای دیگر

دانلود، نصب و راه اندازی پایتون و IDE های قابل استفاده

* دانلود نسخه نرم افزار پایتون از سایت [www.python.org](http://www.python.org)
* نصب آخرین نسخه از زبان پایتون در محیط ویندوز
* معرفی IDE های پرکاربرد زبان برنامه نویسی پایتون

. IDEL Python

. Visual Studio Code

. PyCharm

* راه اندازی و کدنویسی یک برنامه کوتاه

نکته : در صفحه ابتدای نصب Pythone در قسمت پایین تیک Add Pytho 3.12 to PATH را بزنید ، تا مسیر خود Pythone و Script آن در قسمت PATH ویندوز قرار بگیرد. اگر تیک را نزنید می بایست مسیر نرم افزار و Script آن را که معمولا مسیر C:\Users\M\_Shahroodi.IRANET\AppData\Local\Programs\Python\Python312 می باشد را در PATH ویندوز اضافه نمایید . ( روی Mycomputer کلیک راست کرده ، در قسمت Properties ، روی Advaned system settings کلیک کرده ، قسمت Enviroment Variable را با کنید و در قسمت PATH دو مسیر مذکور را اضافه نمایید.)

فصل دوم : شروع برنامه نویسی و آشنایی با اجزای تشکیل دهنده یک برنامه

بخش 1 : کلمات کلیدی زبان پایتون ( Python Key Words ) :

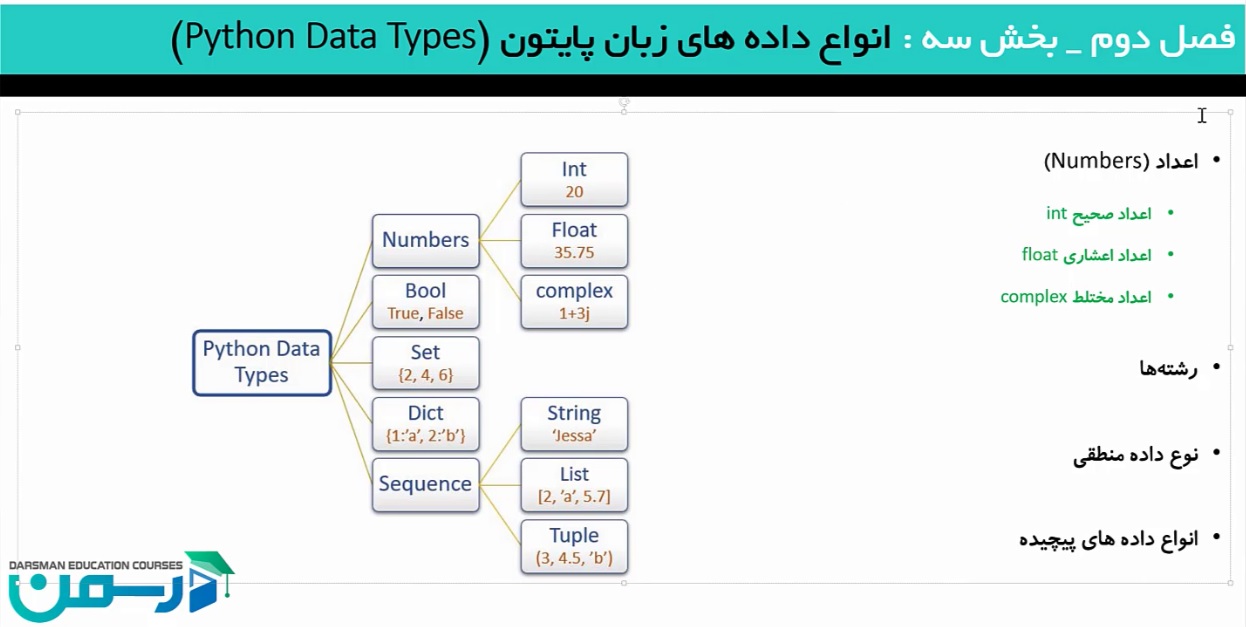


بخش 2 : شناسه ها در زبان پایتون (Python Identifiers) :

کلماتی که داخل زبان نیستن ، اما برنامه نویس نسبت به نیازش در برنامه اضافه میکند identifier یا شناسه گویند. موارد مورد نیاز عبارتند از : نام متغیر - نام ثابت ها – نام توابع – نام ماژول ها – نام کلاس ها و ...

شرایط نامگذاری : با رقم شروع نشود – از بین علامت ها فقط \_ استفاده شود – بهتر است برای متغیرها با فرمت camelCase باشد – بهتر است برای ثابت ها با حروف بزرگ باشد – بهتر ایت برای نام گذاری کلاسها با فرمت PascalCase باشد.

بخش 3 : انواع داده های زبان پایتون (Python Data Type) :



نکته : اعداد صحیح به چهار صورت زیر نمایش داده می شوند :

DEC 10 0..9 num1=65

BIN 2 0..1 num2=0b10110

OCT 8 0..7 num3=0o5207

HEX 16 0..9 A B C D E F num4=0x5AB2

در زبان پایتون زمانیکه متغیر ها با مقادیر عدد صحیح به هر کدام از شکل های فوق تعیین کنیم در خروجی عدد صحیح دسیمال نمایش می دهد.

برای نمایش نوع متغیر از دستور print(type(num1)) استفاده می شود.

نکته: در زبان پایتون می توان متغیر های با مقادیر رشته را به صورت زیر انتخاب کرد ولی تمام حالات زیر به یک شکل نمایش داده می شوند:

Str1= ‘ali reza’ str2= “ali reza” str3=‘‘‘ali reza‘‘‘ str4=“““ali reza“““

Print(str1) print(str2) print(str3) print(str4)

دو حالت str1 و str2 شبیه هم هستند و str3 و str4 نیز شبیه هم ، فرق آنها در این است که در دو حالت اول نمی توان رشته را شکاند یعنی ali باشد و reza را خط بعد برد ولی در دو حالت بعدی می توان این کار را انجام داد.نکته دیگر اینکه چرا ‘ و " که شبیه به هم هستند وجود دارد ، برای اینکه زمانیکه در وسط رشته ای که در متغیر میخواهیم قرار دهیم ‘ یا “ وجود دارد از دیگری برای نمایش استفاده می کنیم به عنوان مثال : str1=’fsfdsg”hgsgs’ str2=”sdwdh’jxhgx”

همچنین برای نمایش کاراکترهای خاص می توان قبل از کاراکتر یک \ قرار داد ، به عنوان مثال :

Str1=”gshffgsxs\”jhshxjsgxsjh\\hjshjsx” 🡪 print(str1) 🡪 gshffgsxs”jhshxjsgxsjh\hjshjsx

بخش 4 : عملگرهای زبان پایتون (Python Operators) :



نکته : // تقسیم صحیح می باشد یعنی قسمت اعشار را نمایش نمیدهد. % یعنی باقیمانده تقسیم دو عدد.

نکته : اگر یک عدد صحیح بر 10 ، تقسیم صحیح (//) شود ، در نمایش عدد یکان آن عدد حذف می شود و با قیمانده (%) همان بر 10 ، در نمایش عدد یکان آن نشان داده می شود.

نکته : علامت \*\* نشان دهنده توان می باشد یعنی عدد اول را به توان عدد دوم برسان.

نکته : در عملگرهای رابطه ای (Relational) حتما True یا False نمایش داده می شود.

در عملگرهای مقداردهی یا ترکیبی (Assignment) :

X+=5 یعنی مقدار X هر چی که هست ، 5 را به آن اضافه کن.

X-=5 یعنی مقدار X هر چی که هست ، 5 را از آن کم کن.

X\*=5 یعنی مقدار X هر چی که هست ، 5 را در آن ضرب کن.

X/=5 یعنی مقدار X هر چی که هست بر 5 تقسیم کن.

X//=5 یعنی مقدار X هر چی که هست بر 5 تقسیم صحیح کن.

X\*\*=5 یعنی مقدار X هر چی که هست به توان 5 برسان.

X%=5 یعنی مقدار X هر چی که هست باقیمانده آن بر 5 را نشان بده.

در عملگرهای منطقی (Logical) :

T and F => F F or T => T not True => False F or not(F) => T

F and T => F F or T => T not False => True

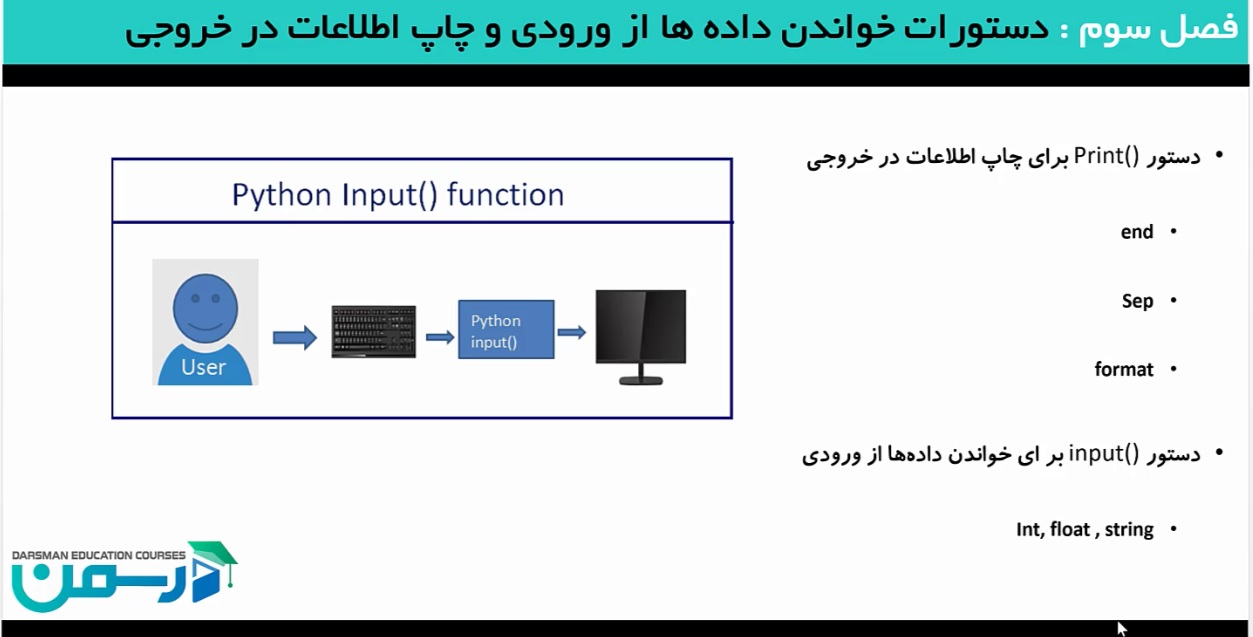
F and F => F F or F => F

T and T => T T or T => T

در عملگرهای بیتی (BitWise) : با این عملگرها به صورت باینری روی متغیرها تغییرات اعمال می شود ، به عنوان مثال اگر x=65 باشد و y=x>>2 باشد ابتدا 65 را به باینری تبدیل کرده و بعد 2 شیفت به راست می دهد و زمانیکه y را چاپ کنیم ، عدد 16 را نشان می دهد.

فصل سوم :

دستورات خواندن داده ها از ورودی و چاپ اطلاعات در خروجی



نکته: دستور print() برای چاپ استفاده می شود که هر مقداری را چاپ می کند به عنوان مثال :

Num1=1000 🡪 print(Num1) print(“Name : “,“Mojtaba“,Num1,True,20)

این دستور پارامتری به نام end= دارد که به طور پیشفرض مقدار آن end=/n است که مکان نما را به خط بعد می برد که می توان آن را تغییر داد:

Print(“Mojtaba“,end=“ – “)

Print(“Ali“,end=“ – “) -----------------> خروجی = Mojtaba - Ali - Mehdi

Print(“Medi“)

همچنین پارامتر دیگری به نام sep دارد که می توان جداکننده برای آن مشخص کرد:

print(“Mojtaba”,”Reza”,”Ali”,sep=’ – ‘) 🡪 Mojtaba - Reza - Ali

در رشته ای که می خواهیم چاپ کنیم اگر کاراکترهای کنترلی وجود داشته باشد و می خواهیم عملا آنها چاپ شوند می توان قبل از کاراترها یک \ قرار داد یا می توان از Raw String استفاده کرد به عنوان مثال :

Print(r”hjsgdghs\tjnsjxsjhxsj\njsakhshj\sjxksxks”) 🡪 خروجی = hjsgdghs\tjnsjxsjhxsj\njsakhshj\sjxksxks

نکته: برای فرمت بندی خروجی یک رشته که می بایست مقادیر متغیرها در رشته نمایش داده شوند به 3 روش زیر می توان عمل کرد :

name=”Mojtaba”

family=”Shahroodi”

age=37

avg=19.75

print(“FirstName : ”,name,”\tLastName : ”,family,”\nAge is : ”,age,”\tAvg is :”,avg)

print(“FirstName : {}\ tLastName : {}\nAge is : {}\tAvg is : {}”.format(name,family,age,avg))

print(f“FirstName : {name}\ tLastName : {family}\nAge is : {age}\tAvg is : {avg}”)

نکته : تابع input() برای دریافت ورودی می باشد که همیشه مقادیر را به صورت str دریافت می کند ، پس اگر ورودی را به صورت نوع های مخلف بخواهیم دریافت کنیم می بایست از توابع مخصوص به آنها استفاده کنیم به عنوان مثال از توابع int() , float() , bool()

num1=float (input(“Enter number : ”))

num2=int(input(“Enter number : ”))

z=num1+num2

print(z)

نکته : هر کاراکتر در کامپیوتر یک کد ASCI دارد به عوان مثال : ‘A‘=65 , ‘a‘=97 ‚ ‘ ‘=32 , ‘0‘=45 , ‘\n‘=13

تابع chr() ، یک عدد می گیرد و به کاراکتر تبدیل می کند.

همچنین تابع ord() برعکس تابع فوق عمل میکند یعنی کاراکتر می گیرد و به عدد تبدیل می کند.

فصل چهارم :

بخش 1 : دستورات شرطی و حلقه های تکرار دذ زبان برنامه نویسی پایتون

* دستورات شرطی if ، if-else ، elif در زبان برنامه نویسی پایتون

• دستورات کنترل روند اجرا برنامه

\* دستور if : به عنوان مثال می بایست به شکل زیر از آن استفاده کرد :

num1=int(input(“Enter width : ”))

num2=int(input(“Enter height : ”))

if num1>0 and num2>0 :

print(num1\*num2) # under control “if”

print(“The End ...“) # out of control “if”

\* دستور if-else :

num1=int(input(“Enter width : ”))

num2=int(input(“Enter height : ”))

if num1>0 and num2>0 :

print(num1\*num2)

else:

print(“Error”)

print(“The End ...“)

\* دستور elif : اگر در یک برنامه ای چندین شرط داشتیم که مطمئن بودیم از بین آنها در لحظه فقط یکی درست است ، از این دستور استفاده میکنیم.

avg=float(input(“Enter AVG : ”))

if 0<=avg and avg<12 :

print(“Mashroot”)

elif 12<=avg and avg<14 :

print(“Zaeif”)

elif 14<=avg and avg<18 :

print(“Motevaset”)

elif 18<=avg and avg<=20 :

print(“Alii”)

else :

print(“Error”)

نکته : وقتی یک شرط داریم می توانیم دستور مربوط به آن را در یک خط بنویسیم : (دستور : شرط if )

اگر دستور نادرست بودن شرط هم بخواهیم داشته باشیم ، می توانیم در یک خط به شکل روبرو بنویسیم: دستور "else" else شرط if دستور "if"

num1=int(input(“Enter width : ”))

num2=int(input(“Enter height : ”))

if num1>0 and num2>0 : print(num1\*num2)

print(num1\*num2) if num1>0 and num2>0 else print(“Error”)

مثالی دیگر در شرط یک خطی:

num1=int(input(“Enter number : ”)) num1=int(input(“Enter number : ”))

if num1<100 : OR x=333 if num1<100 else 999

x=333 print(x)

else :

x=999

print(x)

مسئله : برنامه ای بنویسید که شماره روز و شماره ماه را از کاربر بگیرد و چندمین روز سال را نمایش دهد.

d=int(input("Please Enter day :"))

m=int(input("Please Enter Month :"))

if 1<=m and m<=6 :

   day=(m-1)\*31+d

   print(day)

elif 7<=m and m<=12  :

     day=(m-1)\*30+6+d

     print(day)

else :

    print("Error")

\* دستور match – case : زمانیکه بخواهیم بر اساس مقدار یک متغیر تصمیمات مختلفی بگیریم از این دستور استفاده می کنیم :

num1=int(input("Enter number : "))

match num1 :

    case 1:

        print("ali")

    case 7:

        print("reza")

    case 20:

        print("mojtaba")

    case -3:

        print("ahmad")

    case \_:

        print("Error")

بخش 2 : حلقه های تکرار for و while در زبان برنامه نویسی پایتون

* حلقه های تکرار در زبان پایتون

• حلقه تکرار for و for-else

• حلقه تکرار while و while-else

• تابع range()

* دستورات کنترل روند اجرای حلقه های تکرار

• دستور break

• دستور continue

نکته : برای استفاده از حلقه های تکرار حتما به یک شمارنده نیاز می باشد که برای آن نقطه شروع ، پایان و میزان پرش مشخص شده باشد.

نکته : در حلقه های تکرار برای مشخص کردن تعداد دفعات تکرار می بایست از یک مجموعه استفاده نمود ، که می تواند یک لیست یا یک رشته و غیره باشد همچنین می توان از تابع range() استفاده نمود که می تواند یک مجموعه تولید کند که 3 پارمتر شروع ، پایان (یکی کمتر) و پرش را دارد.برای مثال :range(1,10,+1)

for i in range(1,4,1) => Mojtaba

print(“Mojtaba”) Mojtaba

Mojtaba

for i in range(0,3,1) => Mojtaba\_0

print(f“Mojtaba\_{i}”) Mojtaba\_1

Mojtaba\_2

نکته : اگر در تابع range() میزان پرش را مشخص نکنیم ، خود تابع به صورت پیشفرض 1 در نظر میگیرد. همچنین اگر ابتدای آن را هم مشخص نکنیم 0 در نظر می گیرد ولی حتما باید پاین آن مشخص گردد.

حلقه for معکوس : for i in range(10,0,0-1)

مسئله : دو عدد از ورودی گرفته شود و اعداد مابین آنها با هم ضرب شود.

a=int(input("Enter a : "))

b=int(input("Enter b : "))

res=1

for i in range(a,b+1):

    res\*=i

print(f"result is : {res}")

زمانیکه تعداد دفعات تکرار حلقه برای ما مشخص نباشد ، از while استفاده می کنیم. در این حلقه تا زمانیکه شرط برقرار باشد تکرار می کند.

num=int(input(“Enter Number : ”))

while num>10 :

print(num) 🡪شروع : عدد ورودی ، پایان : شرط و پرش : -1

num-=1

print(num)

print(100\*”-”)

مسئله : یک عدد از کاربر دریافت شود و تعداد ارقام آن چاپ شود.

num=int(input("Enter number : "))

c=0

while num>0 :

    num//=10

    c+=1

print(c)

حلقه های تو در تو :

در این حلقه ها ، حلقه داخلی به ازای هر مقدار حلقه خارجی یک دور کامل می زنند.

# for i in range(1,6):

#     print("Mojtaba Shahroodi")

# for j in range(1,20):

#     print("\*",end='')

for i in range(1,6):

    print("Mojtaba Shahroodi")

    for j in range(1,20):

        if j%2==1 :

            print("\*",end='')

        else :

            print("#",end='')

    print()

Mojtaba Shahroodi

\*#\*#\*#\*#\*#\*#\*#\*#\*#\*

Mojtaba Shahroodi

\*#\*#\*#\*#\*#\*#\*#\*#\*#\*

Mojtaba Shahroodi

\*#\*#\*#\*#\*#\*#\*#\*#\*#\*

Mojtaba Shahroodi

\*#\*#\*#\*#\*#\*#\*#\*#\*#\*

Mojtaba Shahroodi

\*#\*#\*#\*#\*#\*#\*#\*#\*#\*

دستورات break و continue در حلقه ها کاربرد دارند. دستور break باعث شکسته شدن حلقه می شود یعنی خاتمه پیدا کردن اجرای حلقه.

while True:

    a=int(input("Enter a : "))

    if a<0:

        break

    print(a)

for i in range(1,10):

    for j in range(1,21):

        print(j,end=" ")

    if i>5 :

        break

    print()

نکته : دستور continue ما را از حلقه بیرون نمی اندازد ولی باعث می شود که دستورات ادامه خودش در آن مرحله شرط خودش اجرا نشود.

for i in range(1,5) :

    if i==3:

        continue

    print(i)

1

2

4

نکته : در دستور for مانند if می تواند else داشته باشد ولی زمانی اجرا می شود که for تا آخر محدوده مشخص شده اجرا شود و زمانیکه حلقه به هر دلیلی قطع شود ، else اجرا نمی شود به عنوان مثال اگر دستور break در وسط محدوده اجرای حلقه اتفاق بیفتد else اجرا نمی شود ولی continue تاثیری روی آن ندارد.

for i in range(1,5) :

    print(i)

else:

    print("The End...")

1

2

3

4

The End...

نکته: نکته فوق برای دستور while نیز صدق می کند.

i=3

while i>0 :

    print(i)

    i-=1

else:

    print("The End...")

3

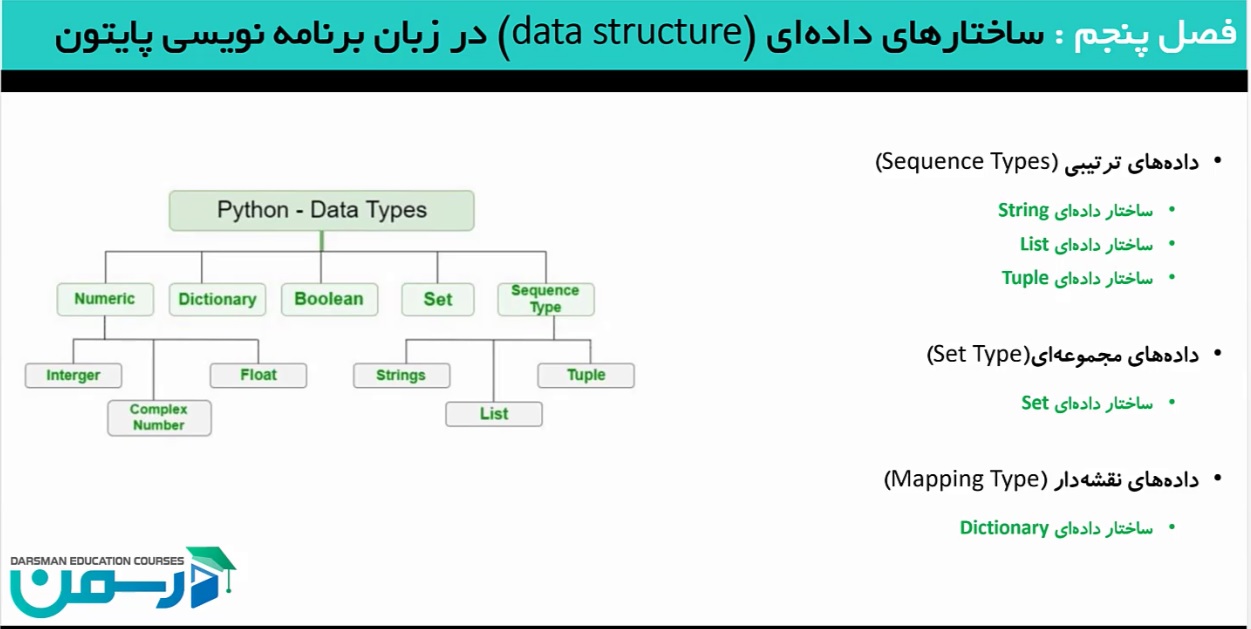
2

1

The End...

فصل پنجم

ساختارهای داده ای (data structure) در زبان برنامه نویسی پایتون



داده های ترتیبی (Sequence Type) :

* ساختار داده ای String : رشته ها در واقع داده هایی هستند شامل انواع کاراکترها. که در داخل "" یا ‘’ قرار می گیرند. همچنین رشته های چند خطی در داخل """ """ یا ‘’’ ‘’’ قرار می گیرند :

Str1=’hsdjcxdjkjsw&^%$#jkkhjjgjjhhkkkk’

Str2=”hsdjcxdjkjsw&^%$#jkkhjjgjjhhkkkk”

Str3=’’’hsdjcxdjkjsw&^%$#jkkhjjgjjhhkkkk’’’

Str4=”””hsdjcxdjkjsw&^%$#jkkhjjgjjhhkkkk”””

نکته: رشته ها به صورت ترتیبی هستند و هر کدام در یک خانه در حافظه قرار می گیرند. حتی میتوان با علامت [] مشخص کرد و یک کاراکتر از رشته را مثلا چاپ کرد.

تابع len() : هر زمان که بخواهیم طول یک رشته را حساب کنیم از تابع len() استفاده می کنیم.

txt1="Mojtaba Shahroodi"

print(txt1)

print(type(txt1))

print(txt1[0])

print(len(txt1))

for i in range(len(txt1)):

    print(txt1[i],end=' ')

for ch in txt1 :

    print(ch,end=' ')

Mojtaba Shahroodi

<class 'str'>

M

17

M o j t a b a S h a h r o o d i

M o j t a b a S h a h r o o d i

* ساختار داده ای List : لیست ها مجموعه داده هایی هستند از هر نوعی که می توان روی آنها هر کاری انجام داد. لیست ها داری نماد [21,”Ali” ,15.7 ,True] هستند.

list1=[22,34,565,767,556,553]

list2=["Mojtaba","Ali","Reza","Sahar","Sara"]

list3=[22.3,455.55,767.55]

list4=[True,False,True,True]

list5=[100,"Mojtaba",5525,888.5,True,"Reza"]

print(list2)

print(type(list2))

for name in list2:

    print(name,end=' ')

print()

for i in range(len(list2)):

    print(list2[i],end=' ')

['Mojtaba', 'Ali', 'Reza', 'Sahar', 'Sara']

<class 'list'>

Mojtaba Ali Reza Sahar Sara

Mojtaba Ali Reza Sahar Sara

نکته: برای اضافه کردن عنصر به لیست می توان از تابع append() استفاده کرد که به انتهای لیست عنصر را اضافه می کند.

numbers=[22,34,565,767,556,553]

numbers.append(6666)

print(numbers)

[22, 34, 565, 767, 556, 553, 6666]

نکته : می توان با تابع extend() به انتهای یک لیست ، اعضای لیست دیگر را اضافه کرد.

list2=["Mojtaba","Ali","Reza","Sahar","Sara"]

numbers=[22,34,565,767,556,553]

numbers.extend([111,222,333,444])

print(numbers)

numbers.extend(list2)

print(numbers)

[22, 34, 565, 767, 556, 553, 111, 222, 333, 444]

[22, 34, 565, 767, 556, 553, 111, 222, 333, 444, 'Mojtaba', 'Ali', 'Reza', 'Sahar', 'Sara']

نکته : با تابع remove() می توان یک عنصر از لیست را حذف نمود.شایان ذکر است عنصری را که اعلام میکنیم و از آن چند تا در لیست داشته باشیم این تابع اولین عنصری را که پیدا می کند حذف می کند.

numbers=[22,34,22,565,22,767,22,556,553]

numbers.remove(22)

print(numbers)

[34, 22, 565, 22, 767, 22, 556, 553]

نکته: با تابع list() می توان یک رشته را به لیست تبدیل کرد.

نکته: با تابع join() می توان عناصر رشته ای یک لیست را به هم چسباند.

مسئله : حرف اول رشته “mehdi” را با حرف بزرگ M عوض کنید. (نکته: در رشته اجازه تغییر هر خانه از رشته وجود ندارد.)

name="mehdi"

temp=list(name)

temp[0]='M'

name=''.join(temp)

print(temp)

Mehdi

* ساختار داده ای Tuple : این ساختار نیز همانند هر دو ساختار قبل عناصر دارای شماره خانه هستند و نماد آن با ( ) نشان داده می شود. تفاوت آن با لیست ها این است که در لیست ها یک کالکشن از داده ها در خود ذخیره می کنند و قابلیت حذف و اضافه و تغییر عناصر وجود دارد اما عناصر تاپل ها ثابت هستند و قابل تغییر نیستند. می توان داخل تاپل ها ، لیست یا تاپل گذاشت ولی قابل تغییر نیستند و هیچکدام از توابع مورد استفاده در قسمت لیست (.join() , .append() , .remove(),…) در اینجا کارایی ندارند.

tuple1=(100,200,300,400,500)

tuple2=("mojta","reza","ahmad","ali","sara")

tuple3=(2000,"mehdi",True,334.5,False)

print(tuple1)

print(tuple[3])

print(type(tuple1))

for item in tuple2:

    print(item,end=' ')

print()

for i in range(len(tuple3)):

    print(tuple3[i],end=' ')

(100, 200, 300, 400, 500)

tuple[3]

<class 'tuple'>

mojta reza ahmad ali sara

2000 mehdi True 334.5 False

تابع tuple() :

مسئله : با وجود اینکه عناصر تاپل ها را نیم توان تغییر داد ، ولی مخواهیم در tuple2 ، تغییرات اعمال کنیم :

tuple2=("mojta","reza","ahmad","ali","sara")

print(tuple2)

templist=list(tuple2)

templist[1]="Reza"

templist.append("negin")

templist.remove("ahmad")

tuple2=tuple(templist)

print(tuple2)

('mojta', 'reza', 'ahmad', 'ali', 'sara')

('mojta', 'Reza', 'ali', 'sara', 'negin')

نکته : همانطور که گفته شد می توان در تاپل ها لیست ها را داشت و بالعکس :

list1=[(23,44,56,66),(55,66,88,33,9)]

tuple1=([100,200,300],[54,65,(2000,3000,4000),544],[74,85,96,52])

print(tuple1)

print(tuple1[1])

print(tuple1[1][2])

print(tuple1[1][2][0])

([100, 200, 300], [54, 65, (2000, 3000, 4000), 544], [74, 85, 96, 52])

[54, 65, (2000, 3000, 4000), 544]

(2000, 3000, 4000)

2000

داده های مجموعه ای (Set Type) :

* ساختار داده ای Set : ساختاری است از مجموعه ها که عناصر تکراری ندارد یعنی اگر به مجموعه ای سه بار ali را بدهید فقط یک بار آن را در نظر می گیرد. و عناصر آن ترتیبی نیست یعنی ترتیب بین اعضا مهم نمی باشد. Set نیز می تواند هر نوع داده ای را بگیرند و با علامت {} مشخص می شود.

برای اضافه کردن عنصر به مجموعه set از تابع .add() و برای حذف کردن عنصر از تابع .remove() استفاده می شود.

set1={12,34,5,6,56,4546,543,5,34,56,5,70}

print(set1)

print(len(set1))

set1.add(1000)

set1.add(5)

print(set1)

set1.remove(5)

print(set1)

{4546, 34, 5, 6, 70, 12, 56, 543}

8

{4546, 34, 5, 6, 70, 1000, 12, 56, 543}

{4546, 34, 6, 70, 1000, 12, 56, 543}

برای استفاده حلقه در مجموعه ها باید به شکل زیر استفاده کرد :

set2={23,435,"Mojtaba",True,34.567,"Mojtaba",2000}

for member in set2:

    print(member,end=' ')

2000 True 34.567 435 Mojtaba 23

نکته : یکی از ابزارهای قدرتمند برای حذف عناصر تکرار در لیست ها و تاپل ها ، set می باشد.

برای حذف عناصر تکراری از لیست بدون استفاده از set :

list1=[12,34,5,6,56,4546,12,12,12,70,543,5,34,56,5,70]

list2=[]

for item in list1:

    if item not in list2:

        list2.append(item)

list1=list2

print(list1)

[12, 34, 5, 6, 56, 4546, 70, 543]

حال با تابع set این کار را انجام میدهیم :

list1=[12,34,5,6,56,4546,12,12,12,70,543,5,34,56,5,70]

set1=set(list1)

list1=list(set1)

print(list1)

[34, 4546, 5, 6, 70, 12, 56, 543]

نکته : در ساختار لیست ها با تابع .count() می توان تعداد عناصر مختلف را شمارش کرد به عنوان مثال در یک لیست چند تا عدد 12 داریم.

list1=[12,34,5,6,56,4546,12,12,12,70,543,5,34,56,5,70]

print(list1.count(12))

4

* داده های نقشه دار (Mapping Type) :

ساختار داده ای Dictionary : ساختار شبیه به ساختار داده ای مجموعه هاست با این تفاوت که علاوه بر قابلیت های مجموعه ها ، امکان دسترسی به عناصر توسط کلید ها را نیز دارد. این ساختار نیز همانند مجموعه ها دارای نماد dict{} است با این تفاوت که عناصر آن دارای کلید می باشد که این کلید می تواند از هر نوعی باشد و همچنین مقادیر نیز می توانند از هر نوعی باشند حتی می توانند لیست یا تاپل یا set باشند. پس dict ها مجموعه هایی هستند که هر عنصر آن داری key:value است و هر کلیدی شاخص دسترسی به یک داده می باشد.

نکته : کلید تکراری در dict ها معنا ندارد ولی می توانیم مقادیر تکراری داشته باشیم.

dict1={'red':24,'blue':80,'yellow':200,'green':60,'blue':250}

print(dict1)

print(dict1['yellow'])

dict1['yellow']=1000

print(dict1)

dict1['black']=500

print(dict1)

{'red': 24, 'blue': 250, 'yellow': 200, 'green': 60}

200

{'red': 24, 'blue': 250, 'yellow': 1000, 'green': 60}

{'red': 24, 'blue': 250, 'yellow': 1000, 'green': 60, 'black': 500}

مثال :

student={'studentId':1,

         'name':'Reza',

         'family':'Ahmadi',

         'age':23,

         'avg':15.67,

         'courses':{'fizik':13.45,'riyazi':17.6,'farsi':19,'arabi':11}

         }

print(student['courses'])

{'fizik': 13.45, 'riyazi': 17.6, 'farsi': 19, 'arabi': 11}

نکته: در استفاده از حلقه ها در این ساختار ، اگر از هیچ تابعی استفاده نشود به طور پیشفرض روی کلیدها حرکت می کند ، اگر از .keys() استفاده شود همانند قبلی روی کلیدها حرکت می کند و اگر از .values() استفاده شود روی مقادیر حرکت می کند و همچنین اگر از تابع .item() استفاده شود، کلید و مقدار آن را یک تاپل درنظر گرفته و روی آنها حرکت می کند.

dic1={'red':24,'blue':80,'yellow':200,'green':60}

for i in dic1:

    print(i,end=' \*\*\* ')

print()

for value in dic1.values():

    print(value,end=' \*\*\* ')

print()

for key in dic1.keys():

    print(key,end=' \*\*\* ')

print()

for item in dic1.items():

    print(item,end=' \*\*\* ')

print()

red \*\*\* blue \*\*\* yellow \*\*\* green \*\*\*

24 \*\*\* 80 \*\*\* 200 \*\*\* 60 \*\*\*

red \*\*\* blue \*\*\* yellow \*\*\* green \*\*\*

('red', 24) \*\*\* ('blue', 80) \*\*\* ('yellow', 200) \*\*\* ('green', 60) \*\*\*

نکته : برای برگرداندن مقدار یک کلید می توانیم از تابع .get() نیز استفاده کنیم.

dic1={'red':24,'blue':80,'yellow':200,'green':60}

print(dic1['yellow'])

print(dic1.get('yellow'))

200

200

فصل ششم

توابع (Functions) و لامبدا در زبان برنامه نویسی پایتوتن :

* ساختار کلی تابع در زبان پایتون
* انواع توابع در پایتون
* فراخوانی توابع در پایتون
* توابع با مقادیر پیشفرض
* توابع با تعداد ورودی های پویا (\*args)
* لامبدا در زبان پایتون

توابع تکه کدهایی هستند که توسط برنامه نویس نوشته می شود که برای آنها یم نام درنظر گرفته می شود و این توابع می توانند یک عمل انجام دهند و برای این عمل ، این توابع می توانند ورودی داشته باشند و یا خروجی ، پس توابع به چهار دسته تقسیم می شوند : 1.توابعی که هم ورودی و هم خروجی دارند 2.توابعی که ورودی ندارند ولی خروجی دارند 3. توابعی که ورودی دارند ولی خروجی ندارند 4.توابعی که نه ورودی دارند و نه خروجی.

در زبان پایتون برای تعریف توابع از کلمه کلیدی def استفاده می شود و بعد در تعریف در برنامه می بایست آن صدا زده شود.

def fun1(num1,num2):

    s=num1+num2

    print(s)

print("------------------")

fun1(20,30)

print("++++++++++++++++++")

------------------

50

++++++++++++++++++

مثال : می خواهیم تابع ای بنویسیم که هر بار صدا زده می شود ، 10 عدد \* به صورت افقی چاپ کند.

def printStar():

    for i in range(10):

        print("\*",end='')

printStar()

\*\*\*\*\*\*\*\*\*\*

نکته : خروجی یک تابع مقادیری نیست که چاپ می شوند بلکه مقداریست که به محل فراخوانی تابع برمیگردد. توابعی خروجی دارند که کلمه کلیدی return را دارند که آن هر مقدار که جلوی آن قرار میگیرد به محل فراخوانی برمیگرداند.

def mul(a,b,c):

    f=a\*b\*c

    return f

print(mul(10,20,30))

6000

نکته : زبان پایتون تنها زبانی است که می تواند بیش از یک مقدار را در فرخوانی توابع برگرداند. و مقادیر بیش از یکی را به صورت tuple بر می گرداند.

def sumAvg(a,b,c,d):

    s=a+b+c+d

    v=s/4

    return s,v

result=sumAvg(10,20,16,7)

print(f"Sum is : {result[0]}")

print(f"Avg is : {result[1]}")

Sum is : 53

Avg is : 13.25

مثال : تابع ای بنویسید که اطلاعات یک فرد را دریافت کند و آن را نمایش دهد.

def showPersonInfo(name,family,address,mobileNumber,age):

    print(f"FullName : {name} {family}")

    print(f"Address : {address}")

    print(f"MobileNumber : {mobileNumber}")

    print(f"Age : {age}")

showPersonInfo('Mojtaba','Shahroodi','Tehran','019124309728',37)

FullName : Mojtaba Shahroodi

Address : Tehran

MobileNumber : 019124309728

Age : 37

حال اگر بخواهیم تابع را طوری فراخوانی کنیم که جای ورودی ها عوض شوند و ترتیب ورودی ها مهم نباشد می بایست به صورت زیر عمل کنیم :

def showPersonInfo(name,family,address,mobileNumber,age):

    print(f"FullName : {name} {family}")

    print(f"Address : {address}")

    print(f"MobileNumber : {mobileNumber}")

    print(f"Age : {age}")

showPersonInfo(name='Mojtaba',family='Shahroodi',age=37,mobileNumber='09124309728',address='Tehran')

نکته : می توان مقادیر پیشفرض برای ورودی های تابع در نظر گرفت که زمانیکه برای یک ورودی مقداری نبود آن در نظر گرفته شود.

def sum(a,b,c=0,d=0):

    return a+b+c+d

print(sum(10,20,30,40))

print(sum(10,20,30))

print(sum(10,20))

100

60

30

نکته : ورودی تابع می تواند هر ساختار داده ای باشد ، مثل لیست ، set ، dict و غیره

def showList(list1):

    for item in list1:

        print(item,end='\t')

    print()

showList([123,43,543,654,85,475])

showList(["Mojtaba","Reza","Ali","Sara","Sahar","Negin"])

123 43 543 654 85 475

Mojtaba Reza Ali Sara Sahar Negin

همچنین خروجی تابع نیز می تواند لیست باشد. مثلا تابع ای بنویسید یک عدد از ورودی بگیرد و یک لیست اعداد تصادفی پر کند.

نکته: در پایتون یکسری ماژول های آماده وجود دارد که این ماژولها از یکسری توابع درست شدند، مانند ماژول random که کافی است با دستور import آن را به برنامه اضافه کنید.

import random

def getRandomList(n):

    list1=[]

    for i in range(n):

        list1.append(random.randint(1,100))

    return list1

tempList=getRandomList(10)

print(tempList)

[42, 77, 89, 35, 51, 49, 19, 71, 89, 39]

توابع با تعداد ورودی های پویا ، اصطلاحا به این ورودی ها packing گفته می شود که ورودی می تواندد هر چیز و هر تعداد باشند

def fun1(\*args):

    for item in args:

        print(item,end='\t')

fun1(12,"ali",True,"reza",2525,665,5445)

12 ali True reza 2525 665 5445

* توابع لامبدا در زبان پایتون (Anonymous-Lambda) :

خیلی مواقع در برنامه نویسی توابعی نوشته می شوند که دستورات خیلی کمی دارند، در حد یک خط یا دو خط ، در این توابع دیگر به تعریف سنتی توابع نیاز نیست ، در واقع یک شکل یا تعریف توابه به صورت خاص می باشد.

def fun(x): fun=lambda x: x\*100

    return x\*100 print(fun(20))

print(fun(20)) ==

2000 2000

مثال :

getFullName=lambda name,family: name+" "+family OR f”{name} {family}”

print(getFullName('Mojtaba','Shahroodi'))

Mojtaba Shahroodi

print((lambda name,family: f"{name} {family}")('Mojtaba','Shahroodi'))

Mojtaba Shahroodi

مثال: برنامه ای بنویسید که چها عمل اصلی را انجام دهد.

def sum(num1,num2): fun = lambda x,y,funcName:funcName(x,y)

    return num1+num2 print(fun(100,40,lambda num1,num2:num1+num2))

def sub(num1,num2): print(fun(100,40,lambda num1,num2:num1-num2))

    return num1-num2 print(fun(100,40,lambda num1,num2:num1\*num2))

def mul(num1,num2): == print(fun(100,40,lambda num1,num2:num1/num2))

    return num1\*num2

def div(num1,num2): 140

    return num1/num2 60

def fun(x,y,funcName): 4000

    print(funcName(x,y)) 2.5

fun(100,40,sum)

fun(100,40,sub)

fun(100,40,mul)

fun(100,40,div)

140

60

4000

2.5

حال با تابع lambda عملیات مرتب سازی را انجام می دهیم. در مثال زیر با استفاده از تایع sorted() عملیات مرتب سازی صورت می گیرد، در قسمت اول که کلید به نمیدهیم قسمت یکسان مقدار را جلو میرود و بر اساس قسمت اول اعداد مرتب سازی را انجام می دهد که نتیجه دلخواه صورت نمی پذیرد ، حال با تابع lambda قسمت کلید را اعداد قرار می دهیم :

names=['mehdi13','mehdi5','mehdi2','mehdi49','mehdi25','mehdi31']

print(sorted(names))

print(sorted(names,key=lambda x:int(x[5:])))

['mehdi13', 'mehdi2', 'mehdi25', 'mehdi31', 'mehdi49', 'mehdi5']

['mehdi2', 'mehdi5', 'mehdi13', 'mehdi25', 'mehdi31', 'mehdi49']

مثال :

dicList=[

    {"name":'mehdi',"age":43},

    {"name":'ali',"age":82},

    {"name":'mehdi',"age":21},

    {"name":'ahmad',"age":39}

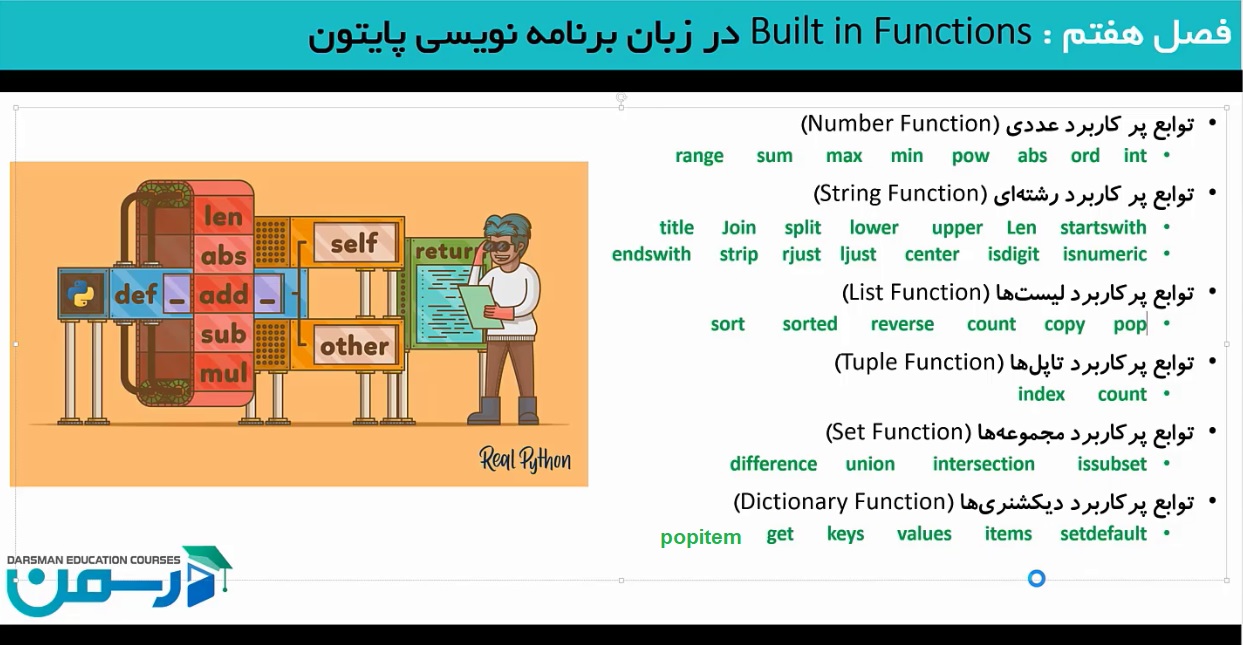
]

print(sorted(dicList,key=lambda dic:dic["name"]))

[{'name': 'ahmad', 'age': 39}, {'name': 'ali', 'age': 82}, {'name': 'mehdi', 'age': 43}, {'name': 'mehdi', 'age': 21}]

فصل هفتم

Built in Functions در زبان برنامه نویسی پایتون



* توابع پرکاربرد عددی (Number Function) :
* تابع int() : برای تبدیل مقدار رشته ای به عددی :

num=int('234')

print(type(num))

<class 'int'>

* تابع ord() : برای تبدیل یک کاراکتر به کد اسکی آن کاراکتر:

str1="Mojtaba Shahroodi"

for ch in str1:

    print(ch,ord(ch))

M 77

o 111

j 106

t 116

a 97

b 98

a 97

32

S 83

h 104

a 97

h 104

r 114

o 111

o 111

d 100

i 105

* تابع sum() برای جمع ، تابع max() بزرگترین عدد مجموعه و تابع min() کوچکترین عدد مجموعه مشخص می کند:

list=[3255,6555,555,66658,332,665,25866,988,8954]

print(sum(list))

print(max(list))

print(min(list))

113828

66658

332

* تابع pow() دو عدد را به توان هم می رساند، پارمتر اول عدد پایه و پارامتر دوم عدد تابع می شود :

print(pow(3,8))

6561

* تابع abs() برای قدر مطلق عدد استفاده می شود :

print(abs(120)) x=23

print(abs(-120)) y=90

120 print(abs(x-y))

120 print(abs(y-x))

* تابع range() ، این تابع می تواند یک محدوده عددی را تولید کند.

list1=list(range(1,30,3))

print(list1)

[1, 4, 7, 10, 13, 16, 19, 22, 25, 28]

* توابع پر کاربرد رشته ای (String Funcation) :
* تابع title() ، اگر از تابع روی یک رشته استفاده کنید حرف اول هر کلمه را به حروف بزرگ تبدیل می کند :

name="mojtaba"

family="shahroodi"

print(name.title(),family.title())

Mojtaba Shahroodi

* تابع len() که طول رشته را بدست می آورد.
* تابع upper() تمام حروف رشته را به حرف بزرگ و تابع lower() برعکس تمام حروف رشته را به حروف کوچک تبدیل می کند.

name="Mojtaba Shahroodi"

print(name.upper())

print(name.lower())

MOJTABA SHAHROODI

mojtaba shahroodi

* تابع split() برای شکستن یک رشته بر حسب آن چیزی که به عنوان پارامتر به آن می دهیم استفاده می شود و لیستی از رشته ها می دهد.

str1="hcgch kjshcsd dsosuds oocshy oocxsiju sosuds"

print(str1.split("h"))

['', 'cgc', ' kjs', 'csd dsosuds oocs', 'y oocxsiju sosuds']

* حال لیستی از رشته ها داریم که می خواهیم برعکس تابع قبل آنها را به هم بچسبانیم ، برای این کار از تابع join() استفاده می کنیم. برای استفاده از آن ابتدا باید کارکتری که بر حسب آن رشته ها به هم چسبانده شوند را مشخص می کنیم.

list1=["mojtaba","reza","ali","sahar","sara","negin"]

print('-'.join(list1))

mojtaba-reza-ali-sahar-sara-negin

* تابع startswith() و endswith() برای چکینگ استفاده می شوند و یک bool برمیگردانند. در تابع startswith() اگر یک رشته با آن چیزی که به عنوان پارامتر مشخص می شود شروع شده باشد True و اگر نه False بر می گرداند. تابع endswith() نیز انتهای رشته را چک می کند.

str1="swdcfrfrjhh dsdjksh kjldjc djdcyhesd jhgjhsggd sju"

print(str1.startswith('swd'))

print(str1.startswith('mehdi'))

print(str1.endswith('sju'))

print(str1.endswith('mehdi'))

True

False

True

False

* توابع strip() ، rjust() ، ljust() و center() برای نرمال سازی رشته استفاده می شود. تابع strip() برای حذف space های ابتدا و انتهای رشته استفاده می شود.

تابع ljust() برای آنکه یک رشته داریم و میخواهیم مثلا در یک فضای 10 کاراکتری نمایش دهیم و از سمت چپ تراز شود.

تابع rjust() همانند قبلی است با این تفاوت که رشته را از سمت راست تراز می کند.

تابع center() نیز همانند قبلی ها می باشد با این تفاوت که رشته در وسط قرار می گیرد

str1="     mehdi@yahoo.com        "

print("\*"+str1+"\*")

print("\*"+str1.strip()+"\*")

str2="mehdi"

print(str2+"\*")

print(str2.ljust(10,' ')+'\*')

print(str2.rjust(10,' ')+'\*')

print("\*"+str2.center(10,' ')+'\*')

\* mehdi@yahoo.com \*

\*mehdi@yahoo.com\*

mehdi\*

mehdi \*

mehdi\*

\* mehdi \*

* برای آنکه چک کنید یک رشته عدد می باشد یا خیر از توابع isdigit() و isnumeric() استفاده می شود. تابع isdigit() یعنی رشته تشکیل شده فقط از رقم ها (0,1,2..,9) یا کدهای اسکی رقمی و تابع isnumeric() هم یعنی رشته از اعداد یا اعداد unicode تشکیل شده است. و خروجی هر دو تابع bool می باشد.

print('2145'.isdigit(),'2145'.isnumeric())

print('21a45'.isdigit(),'21a45'.isnumeric())

print('21/45'.isdigit(),'21/45'.isnumeric())

print('21.45'.isdigit(),'21.45'.isnumeric())

print('½'.isdigit(),'½'.isnumeric())

print('¼'.isdigit(),'¼'.isnumeric())

True True

False False

False False

False False

False True

False True

* توابع پر کاربرد لیست ها(List Funcation) :
* تابع sort() : مقادیر لیست را از کوچک به بزرگ مرتب می کند. این تابع لیست را درون خودش مرتب می کند.

list1=[324,45,5,6758,654,534,67,4,63,4]

list1.sort()

print(list1)

[4, 4, 5, 45, 63, 67, 324, 534, 654, 6758]

* تابع sorted() : این تابع نیز مانند قبلی مقادیر لیست را مرتب می کند با این تفاوت که خود لیست را تغییر نمی دهد و در لیست دیگر می توان نمایش داد.

list1=[324,45,5,6758,654,534,67,4,63,4]

list2=sorted(list1)

print(list1)

print(list2)

[324, 45, 5, 6758, 654, 534, 67, 4, 63, 4]

[4, 4, 5, 45, 63, 67, 324, 534, 654, 6758]

* تابع reverse() ، با این تابع می توان یک لیست را برعکس کرد.

list1=[324,45,5,6758,654,534,67,4,63,4]

list1.reverse()

print(list1)

[4, 63, 4, 67, 534, 654, 6758, 5, 45, 324]

* تابع count() ، این تابع می تواند تعداد تکرار یک عنصر در لیست را شمارش کند.

list1=[324,45,5,6758,654,534,67,4,63,4]

print(len(list1))

print(list1.count(4))

10

2

* تابع copy() ، این تابع یک لیست را درون لیست دیگر کپی می کند.

list1=[324,45,5,6758,654,534,67,4,63,4]

list2=list1.copy()

print(list2)

[324, 45, 5, 6758, 654, 534, 67, 4, 63, 4]

* تابع pop() ، این تابع آخرین عنصر یک لیست را برمیدارد و حذف می کند.

list1=[324,45,5,6758,654,534,67,4,63,4]

print(list1.pop())

print(list1)

4

[324, 45, 5, 6758, 654, 534, 67, 4, 63]

مثال:

list1=[324,45,5,6758,654,534,67,4,63,4]

while len(list1)>0 :

    x=list1.pop()

    print(x)

print(list1)

4

63

4

67

534

654

6758

5

45

324

[]

* توابع پرکاربرد تاپل ها :
* تابع count() ، همانند لیست ، مشخص می کند یک عنصر از تاپل چندبار تکرار شده است.
* تابع index() ، این تابع شماره یک عنصر از تاپل را برمیگرداند. در واقع عمل جستجو را انجام می دهد.و اگر یک عنصر مورد جستجو در تاپل وجود نداشته باشد خطا بر می گرداند.

tuple1=(324,45,5,6758,654,534,67,4,63,4)

print(tuple1.count(4))

print(tuple1.index(534))

2

5

* توابع پرکابرد مجموعه ها (Set Funcation) :
* تابع difference() ، این تابع برای مقایسه دو مجموعه یا اختلاف آنها استفاده می شود.

set1={12,45,78,345,897}

set2={15,68,12,100,78}

temp1=set1.difference(set2)

temp2=set2.difference(set1)

print(temp1)

print(temp2)

{345, 897, 45}

{100, 15, 68}

* تابع union() ، این تابع اجتماع دو مجموعه را انجام می دهد.
* تابع intersection() ، این تابع عمل اشتراک را بین دو مجموعه انجام می دهد.
* تابع issubset() ، این تابع نشان می دهد یک مجموعه زیر مجموعه ، مجموعه دیگری هست یا خیرو خروجی True یا False برمیگرداند.

set1={12,45,78,345,897}

set2={15,68,12,100,78}

set3={15,12}

set4=set1.union(set2)

set5=set1.intersection(set2)

print(set4)

print(set5)

print(set1.issubset(set2))

print(set3.issubset(set2))

{897, 100, 68, 12, 45, 78, 15, 345}

{12, 78}

False

True

* توابع پرکاربرد دیکشنری (Dictionary Funcation) :
* تابع get() ، برای واکشی مقادیر کلیدها می توانیم از ایت تابع استفاده کنیم. با این تابع اگر مقدار کلیدی که وجود ندارد را واکشی کنیم خطا بر نمیگرداند و None را برمیگرداند.
* تابع key() ، لیستی از کلیدهای دیکشنری را نشام می دهد.
* تابع values() ، لیست مقادیر کلیدها را برمیگیرداند.
* تابع items() ، لیست کلیدها با مقادیرشان را بر میگرداند.
* تابع setdefault() ، این تابع برای زمانی می تواند استفاده شود که اگر یک کلیدی را در دیکشنری نداشتیم مثل color در تابع get ، مثلا مقدار Red را برای آن برگرداند. اگر کلیدی وجود داشته باشد این تابع برای آن پیشفرض را نمیاورد.

dic1={"name":"mojtaba","family":"shahroodi","age":37,"avg":19.87}

print(dic1)

print(dic1.get("name"))

print(dic1.get("color"))

print(dic1.keys())

print(dic1.values())

print(dic1.items())

dic1.setdefault("color","Red")

print(dic1.get("color"))

{'name': 'mojtaba', 'family': 'shahroodi', 'age': 37, 'avg': 19.87}

mojtaba

None

dict\_keys(['name', 'family', 'age', 'avg'])

dict\_values(['mojtaba', 'shahroodi', 37, 19.87])

dict\_items([('name', 'mojtaba'), ('family', 'shahroodi'), ('age', 37), ('avg', 19.87)])

Red

مثال : می خواهیم تعداد هر کاراکتر در یک رشته را پیدا کنیم.

text1="mojtaba ali reza mehdi mohammad sara saeed saman"

dicCountChar={}

for ch in text1:

    dicCountChar.setdefault(ch,0)

    dicCountChar[ch]+=1

print(dicCountChar)

dicNew=sorted(dicCountChar)

print(dicNew)

{'m': 6, 'o': 2, 'j': 1, 't': 1, 'a': 11, 'b': 1, ' ': 7, 'l': 1, 'i': 2, 'r': 2, 'e': 4, 'z': 1, 'h': 2, 'd': 3, 's': 3, 'n': 1}

[' ', 'a', 'b', 'd', 'e', 'h', 'i', 'j', 'l', 'm', 'n', 'o', 'r', 's', 't', 'z']

فصل هشتم

شی گرایی (OOP) در زبان برنامه نویسی پایتون

* آشنایی کامل با مفهوم شی گرایی
* تعریف کلاس و اعضای درون کلاس
* اعضای داده ای کلاس
* اعضای تابعی کلاس
* معرفی تابع سازنده کلاس (\_\_int\_\_())
* معرفی تابع \_\_str\_\_() در کلاس
* معرفی class variable ها
* تعریف نمونه های کلاس و استفاده از کلاس
* آشنایی اولیه با کپسوله سازی ، چندریختی و وراثت در کلاس ها
* آشنایی با کلاس های Enum

تا قبل از این برنامه های ما به صورت ساختیافته بود به صورتیکه برنامه مبتنی بر توابع و عمل ها بود در مفهوم شی گرایی برنامه مبتنی بر داده می باشد . در برنامه نویسی شی گرا ابتدا از کارفرما سوال میکنیم که قرار است با چه داده هایی کار کنیم؟ بعد از اینکه مشخص شده چه داده هایی در پروژه ما باید ذخیره شود ، سوال بعدی این است که ، چه کارهایی روی این داده ها باید انجام شود؟

ولی در برنامه نویسی ساختیافته فقط سوال این بود که چه کارهایی باید انجام شود؟

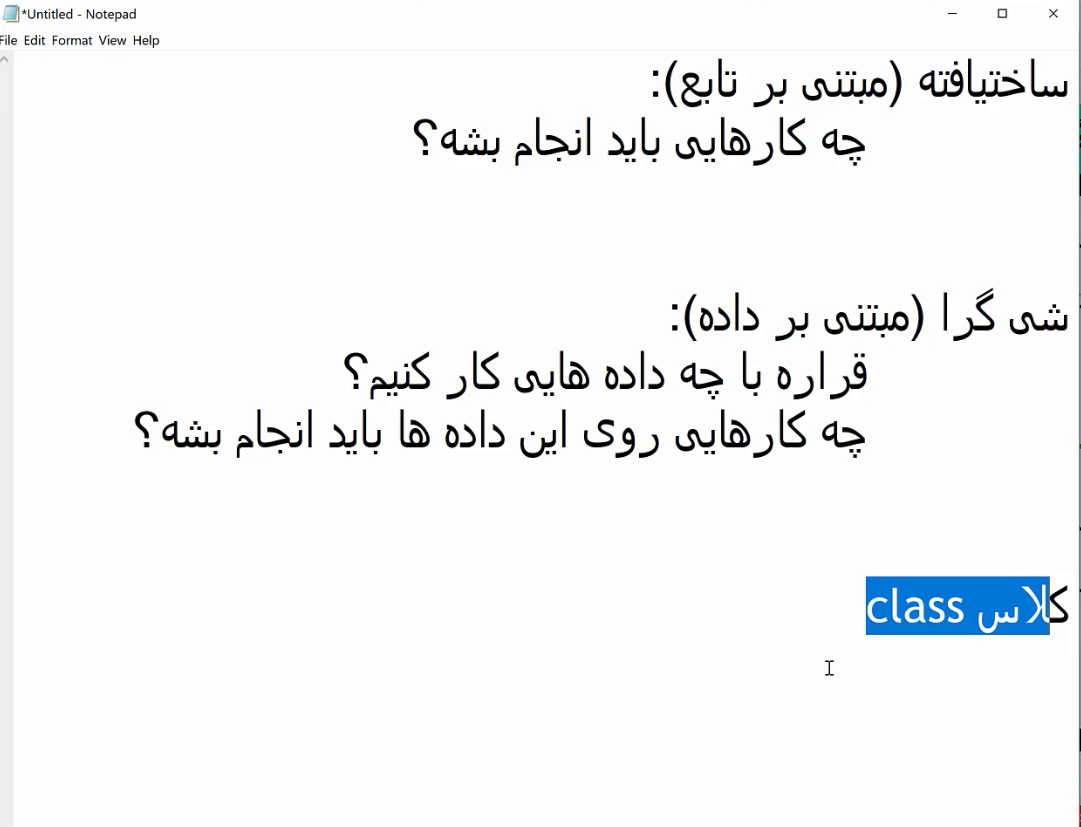
ما برای پیاده سازی برنامه شی گرایی ، نیاز به نوشتن کلاس (class) داریم. این کلاس بدین معناست که کسانی که در این کلاس هستند یکسری ویژگی های مشترک دارند. به عنوان مثال در دانشگاه ، موجودیت ها یا شی های دانشجو ، استاد ، درس و گروه آموزش وجود دارند. در اینجا دانشجوها یکسری صفت های مشترک دارند پس کلاس دانشجو را تشکیل می دهند و به همین ترتیب کلاس های استاد ، درس و گروه آموزش نیز تشکیل می شوند.

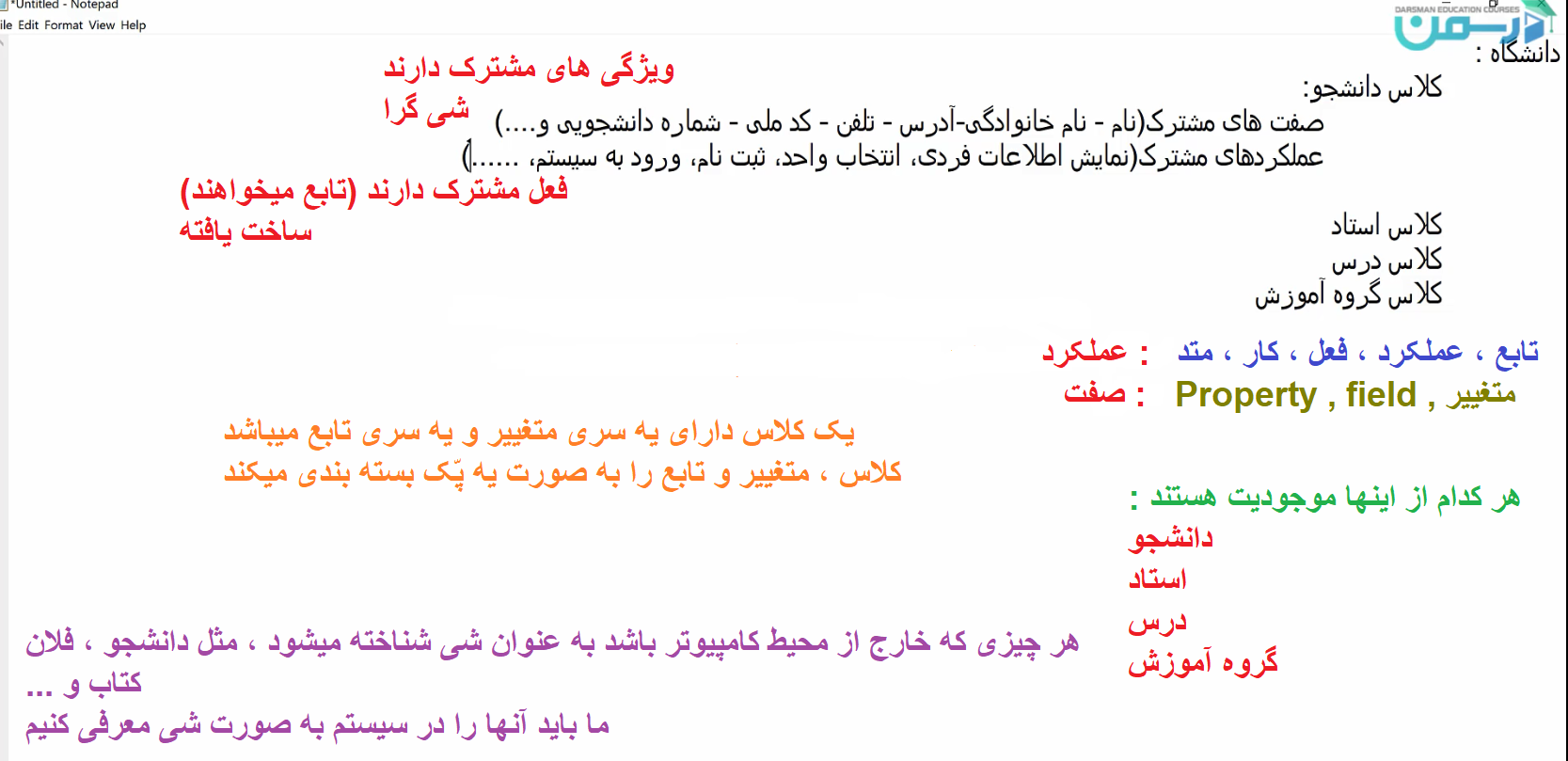
برای نوشتن کلاس ، دو چیز مهم می باشد : صفت های مشترک و عملکردهای مشترک. به عنوان مثال برای کلاس دانشجو :

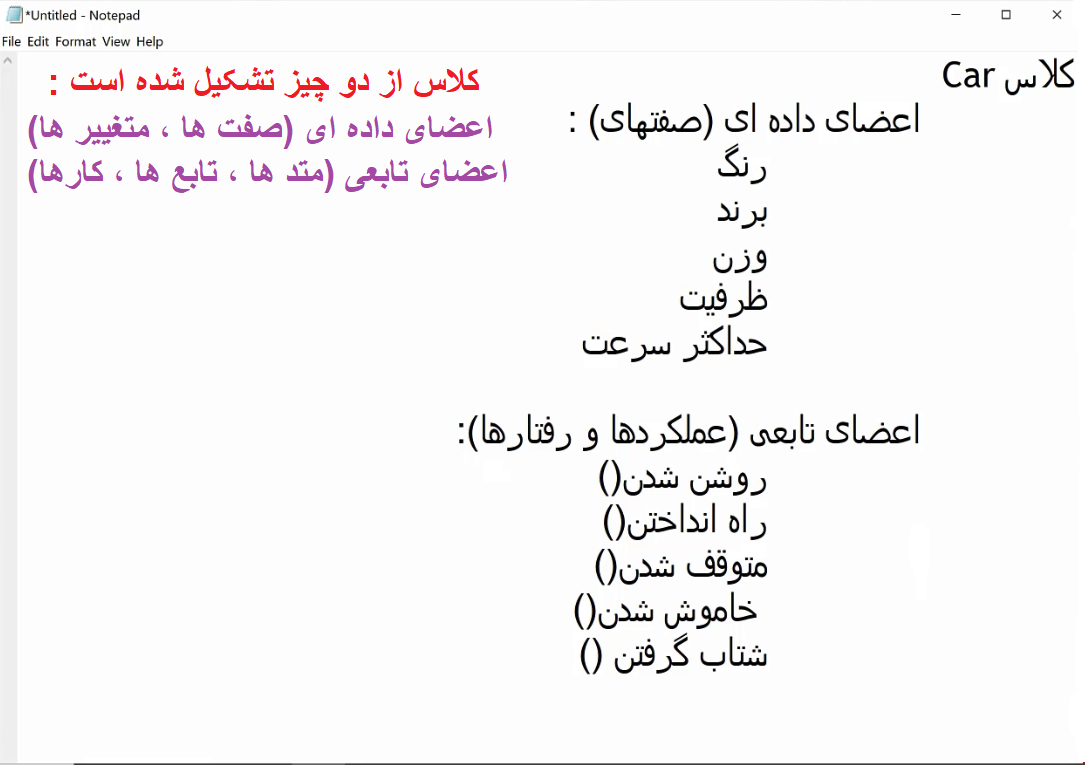
صفت های مشترک (نام – نام خانوادگی – نام پدر – آدرس – تلفن – کدملی – شماره دانشجویی-...)

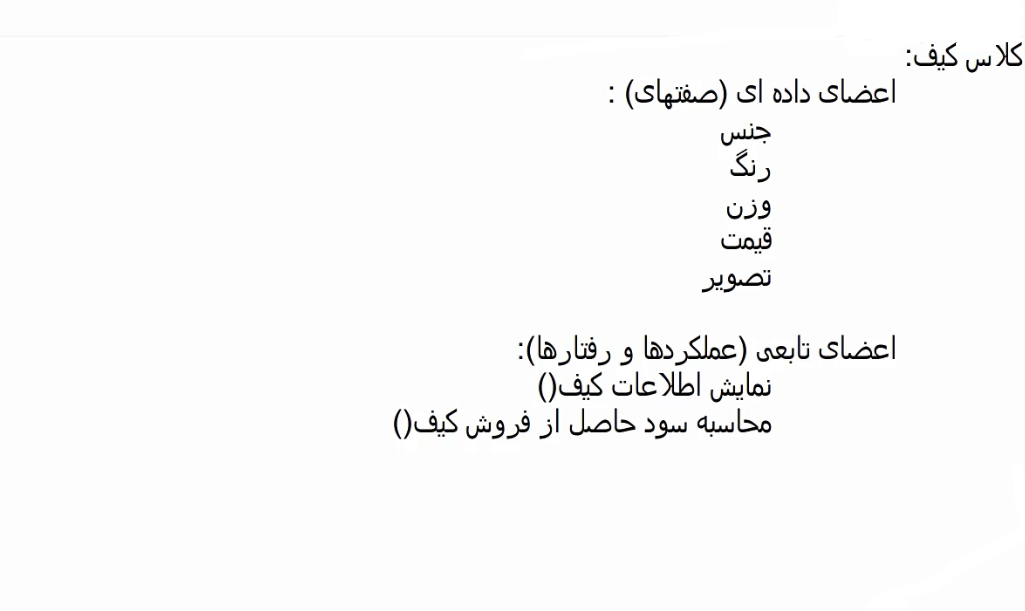
عملکردهای مشترک (نمایش اطلاعات فردی – انتخاب واحد – ثبت نام –ورود به سیستم - ...)

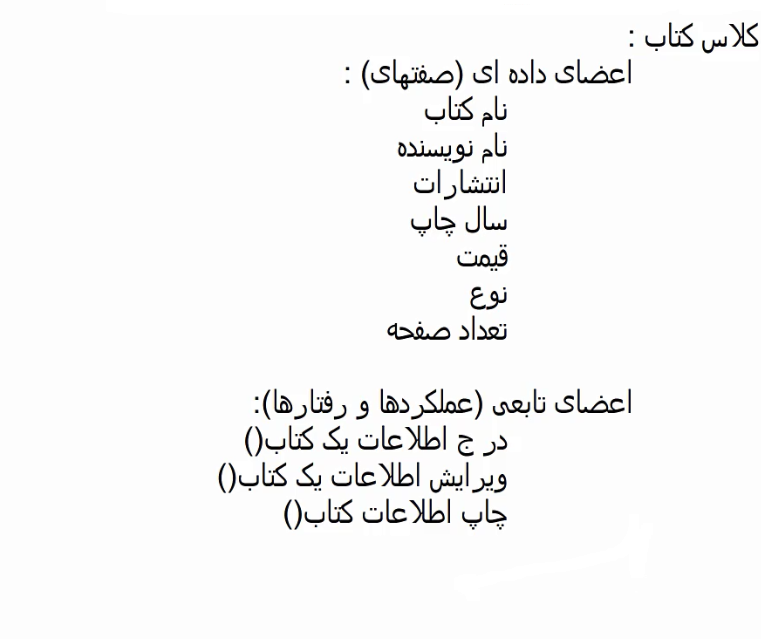
نکته : عملکردها در پایتون همان توابع هستند. و صفت نیز همان متغیرها هستند.











نکته : کلاس ها یک نوع داده جدید هستند که توسط برنامه نویس تعریف می شوند و از آن لحظه به بعد درون پروژه وجود دارند و ما می توانیم از آنها نمونه تعریف کنیم. مثلا برای کلاس car بعد تعریف می توان نمونه های زیر را تعریف کرد :

Class Car:

color=”red”

brand=”prid”

weight=2800

capacity=4

maxSpeed=220

def startEngine():

print(“Start Car”)

def stopEngine():

print(“Stop Car”)

def accelerate():

print(“Accelerate Car”)

car1=Car()

car2=Car()

car3=Car()

اکنون برای مثال فوق مقادیر برای هر سه نمومه car1,2,3 یکسان می باشد یعنی مثلا رنگ هر سه قرمز می باشد ، حال اگر بخواهیم مثال فوق را منطقی تر پیش ببریم می بایست هر نمونه که از کلاس car استفاده می کند مقادیر مجزا داشته باشد برای این کار نیاز به متد یا تابع سازنده می باشد. متد سازنده یکی از اعضای کلاس می باشد، متد سازنده با نام initial یا init معرفی می شود و این متد برای مقداردهی اولیه فیلدهای کلاس استفاده می شود.مثلا برای مثال فوق صفتهای color,brand,weight,capacity,maxSpeed به جای اینکه در بدنه کلاس باشد در تابع init قرار می گیرند و مقدایر را به عنوان ورودی به تابع init می دهیم.

نکته : از آنجا که همه اشیا یا نمونه ها از توابع کلاس به صورت اشتراکی استفاده می کنند و مقادیر نمونه ها مشخص می باشد ولی برای آنکه مشخص شود که از توابع کدوم شی دارد استفاده می کند مشخص نیست یه همین علت همیشه وقتی تابعی را در یک کلاس تعریف می کنیم ، ابتدا یک ورودی به نام self بهش می دهیم ، این self کسی است که باعث فراخوانی توابع شده است یعنی برای self نام نمونه ارسال می شود همچنین مقادیر اولیه در تابع سازنده نیز می بایست با self. شروع شوند.

نکته : توابع سازنده در لحظه تعریف نمونه کلاس به صورت اتوماتیک فراخوانی می شوند و چون در کلاس تابع سازنده داریم پس حتما باید نمونه مقادیر خود را ارسال کند.

نکته : در کلاس ها برای دسترسی به اعضای هر نمونه از کلاس ، اسم نمونه به اضافه "." و اسم صفت را می نویسیم.

نکته: اشیا به خودی خود قابل نمایش نیستند یعنی نمی توانی بنویسیم : print(car1) اما فیلد های آنها را می توان نمایش داد یا از توابع آنها استفاده کرد.

class Car:

    def \_\_init\_\_(self,color,brand,weight,capacity,maxSpeed):

        self.color=color

        self.brand=brand

        self.weight=weight

        self.capacity=capacity

        self.maxSpeed=maxSpeed

    def startEngine(self):

        print("Start Car")

    def stopEngine(self):

        print("Stop Car")

    def accelerate(self):

        print("Accelerate Car")

car1=Car("blue","benz",3400,5,260)

car2=Car("black","mvm",1790,4,180)

car3=Car("red","prid",1900,6,360)

print(car1.color,car1.brand,car1.weight,car1.capacity,car1.maxSpeed)

print(car2.color,car2.brand,car2.weight,car2.capacity,car2.maxSpeed)

print(car3.color,car3.brand,car3.weight,car3.capacity,car3.maxSpeed)

car1.startEngine()

car2.stopEngine()

car3.accelerate()

blue benz 3400 5 260

black mvm 1790 4 180

red prid 1900 6 360

Start Car

Stop Car

Accelerate Car

معرفی تابع \_\_str\_\_() : قبلا زمانیکه خود اشیا (car1,2,3) را می خواستیم نمایش دهیم با خطای اینکه این اشیا object هستند مواجه می شدیم حتی تابع str() نیز روی اشیا جواب نمی داد ، برای آنکه بتوانیم خود اشیا را نمایش دهیم می بایست در کلاس تابع \_\_str\_\_() را تعریف کنیم :

class Car:

    def \_\_init\_\_(self,color,brand,weight,capacity,maxSpeed):

        self.color=color

        self.brand=brand

        self.weight=weight

        self.capacity=capacity

        self.maxSpeed=maxSpeed

    def \_\_str\_\_(self):

        return f"Brand : {self.brand}\tcolor :{self.color}\tMax Speed :{self.maxSpeed}\t"

    def startEngine(self):

        print("Start Car")

    def stopEngine(self):

        print("Stop Car")

    def accelerate(self):

        print("Accelerate Car")

car1=Car("blue","benz",3400,5,260)

car2=Car("black","mvm",1790,4,180)

car3=Car("red","prid",1900,6,360)

print(str(car1))

print(car1)

print(car2)

print(car3)

Brand : benz color :blue Max Speed :260

Brand : benz color :blue Max Speed :260

Brand : mvm color :black Max Speed :180

Brand : prid color :red Max Speed :360

معرفی کلاس variable ها : ما عملا سه دسته variable یا همان متغیر در کلاس داریم :

* یک از آنها متغیرهای معمولی است که به آنها instance variable گفته می شود که متغیرهایی می باشند که متعلق به شی می باشند ( مثل self.color ).
* دیگری class variable ها می باشند ، زمانیکه شما یک متغیر در کلاس تعریف می کنید و نه در تابع و همچنین کلمه self. برای آن استفاده نمی شود بدین معناست این متغیر به صورت سراسری بوده و برای همه یکسان استفاده می شود. در واقع این متغیر متعلق به یک شی خاص نمی باشد و متعلق به همه است. برای دستری به این متغیر ها هم می توان از طریق اشیا این کار را انجام داد و هم از طریق نام کلاس ولی پیشنهاد می گردد از طریق نام کلاس به آنها دسترسی پیدا کرد. برای مثال فوق می خواهیم یک شمارنده داشته باشیم که هر شی که ساخته شد یکی به آن اضافه شود:

class Car:

    count=0

    def \_\_init\_\_(self,color,brand,weight,capacity,maxSpeed):

        self.color=color

        self.brand=brand

        self.weight=weight

        self.capacity=capacity

        self.maxSpeed=maxSpeed

        Car.count+=1

    def \_\_str\_\_(self):

        return f"Brand : {self.brand}\tcolor :{self.color}\tMax Speed :{self.maxSpeed}\t"

    def startEngine(self):

        print("Start Car")

    def stopEngine(self):

        print("Stop Car")

    def accelerate(self):

        print("Accelerate Car")

print(Car.count)

car1=Car("blue","benz",3400,5,260)

car2=Car("black","mvm",1790,4,180)

car3=Car("red","prid",1900,6,360)

print(Car.count)

0

3

آشنایی اولیه با کپسوله سازی ، چند ریختی و وراثت در کلاس ها :

* کپسوله سازی یا Encapsulation : کلاس ها باید طوری طراحی شوند که اعضای داده ای مهم آنها در محدوده کلاس از آنها محافظت شود و از بیرون نتوانیم به راحتی یا به صورت مستقیم به آنها دسترسی داشته باشیم و از بیرون کلاس از طریق یک واسطه به آن اعضا داده ای خاص دسترسی داشته باشیم. و برای خصوصی کردن اعضا داده ای که دیگر بیرون کلاس اجازه دسترسی و تغییر آن نباشد کافیست قبل از فیلد "\_\_" قرار دهیم.

در کل سه سطح دسترسی برای اعضا داده ای داریم : 1. خصوصی یا private که قبل از فیلد داده ای "\_\_" قرار می گیرد. 2. محافظت شده یا protected که قبل از فیلد داده ای "\_" قرار می گیرد. 3. عمومی یا public که قبل از فیلد داده ای چیزی قرار نمی گیرد.

نکته : برای توابع هم اگر بخواهیم از بیرون کلاس به آن دسترسی نداشته باشیم کافیست "\_\_" قبل از نام تابع قرار دهیم و دیگر مقدار self نیز برای آن نیاز نیست.

class Person:

    def \_\_init\_\_(self,name,family,age):

        self.\_\_name = name

        self.\_\_family = family

        self.age = age

    def \_\_str\_\_(self):

        return self.\_\_name+"\t"+self.\_\_family+"\t"+str(self.age)

    def \_\_fun1():

        print("Hello")

#-----------------------------------------------------------------

p1=Person("Mojtaba","Shahroodi",37)

print(p1)

p1.\_\_name="Ali"

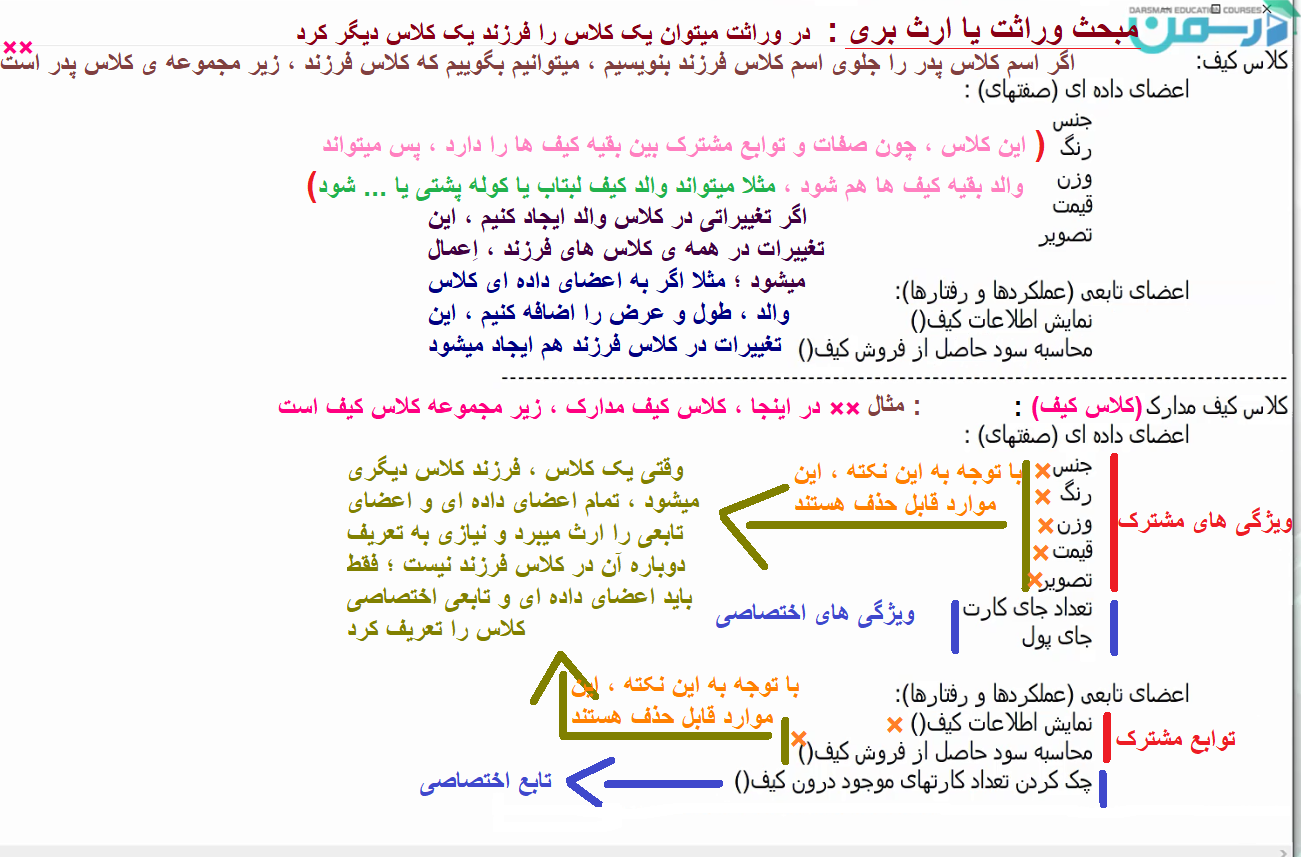
p1.age=40

print(p1)

Mojtaba Shahroodi 37

Mojtaba Shahroodi 40

* وراثت یا Inheritance :



در وراثت کلاس هایی که صفات و توابع مشترک دارند ، یک کلاس والد نوشته می شود و صفات و توابع مشترک به آن داده می شود و در جلوی تعریف کلاس های فرزند نام کلاس والد قرار داده می شود ، نکته مهم اینجاست که اگر کلاس والدی دارای تابع سازنده با ورودی بود ، باید در ابتدای هر تابع سازنده فرزندان ، نیاز های والد را تامین کنیم که این کار به این شکل است که با کلمه super() تابع سازنده والد را صدا می زنیم.

class Person:

    def \_\_init\_\_(self,name,family,phone,address):

        self.name = name

        self.family = family

        self.phone = phone

        self.address = address

    def showInfo(self):

        print(self)

#------------------------------------------------------------------------------------------------------------------

class Student(Person):

    def \_\_init\_\_(self,shNumber,name,family,phone,address):

        super().\_\_init\_\_(name,family,phone,address)

        self.shNumber = shNumber

    def \_\_str\_\_(self) :

        return str(self.shNumber) + " " + self.name + " " + self.family + " " + self.phone + " " + self.address

#------------------------------------------------------------------------------------------------------------------

class Teacher(Person):

    def \_\_init\_\_(self,teacherCode,name,family,phone,address):

        super().\_\_init\_\_(name,family,phone,address)

        self.teacherCode = teacherCode

    def \_\_str\_\_(self) :

        return str(self.teacherCode) + " " + self.name + " " + self.family + " " + self.phone + " " + self.address

#------------------------------------------------------------------------------------------------------------------

class Employee(Person):

    def \_\_init\_\_(self,employeeId,name,family,phone,address):

        super().\_\_init\_\_(name,family,phone,address)

        self.employeeId = employeeId

    def \_\_str\_\_(self) :

        return str(self.employeeId) + " " + self.name + " " + self.family + " " + self.phone + " " + self.address

#------------------------------------------------------------------------------------------------------------------

s1=Student(1200,'Ali','Rezaei','0912000000','Tehran')

t1=Teacher(11350,'Mojtaba','Shahroodi','0912111111','Tehran')

emp1=Employee(100,'Reza','Akbari','0912010101','Arak')

t1.showInfo()

s1.showInfo()

emp1.showInfo()

11350 Mojtaba Shahroodi 0912111111 Tehran

1200 Ali Rezaei 0912000000 Tehran

100 Reza Akbari 0912010101 Arak

* چند ریختی یا polymorphic : اگر درون یک برنامه یا یک کلاس پایتونی تابعی داشته باشیم که به شکل های مختلف یا به ریخت های مختلف بشود از آن استفاده کرد عملا polymorphic را پیاده سازی کرده ایم. به عنوان مثال در یک برنامه یک تابع sum داریم که جمع دو عدد را انجام می دهد ، حال اگر طوری این تابع را پیاده سازی کنیم که مثلا سه عدد صحیح را نیز با هم جمع کند و یا برای دو رشته نیز جواب دهد پس ما یک polymorphic در سطح برنامه داریم. و همچنین اگر در سطح وراثت ما بتوانیم توابع همنامی داشته باشیم بین کلاس والد و فرزند باز هم polymorphic پیاده سازی شده است. پس اگر یک تابع داشته باشیم که به نوع های مختلف بشود از آن استفاده کرد polymorphic در سطح برنامه داریم از نوع overloading .

def sum(a,b,c=0):

    return a+b+c

print(sum(10,20,30))

print(sum(10,20))

60

30

حال در سطح وراثت بررسی می کنیم : زمانیکه در کلاس والد یک تابع داشته باشیم که همنام آن تابع در کلاس فرزند نیز وجود داشته باشد در هنگام فراخوانی تابع خود همان کلاس فرزند فراخوانی می شود که به آن polymorphic از نوع overriding می گویند :

class Person:

    def \_\_init\_\_(self,name,family,age):

        self.name = name

        self.family = family

        self.age = age

    def \_\_str\_\_(self):

        return self.name+" "+self.family+" "+self.age

    def show(self):

        print("Person ...")

#--------------------------------------------------------------

class Student(Person):

    def \_\_init\_\_(self,studentId,name,family,age):

        Person.\_\_init\_\_(self,name,family,age)

        self.studentId = studentId

    def show(self):

        print("Student ...")

#--------------------------------------------------------------

class Teacher(Person):

    def \_\_init\_\_(self,teachertId,name,family,age):

        Person.\_\_init\_\_(self,name,family,age)

        self.teachertId = teachertId

#--------------------------------------------------------------

s1=Student(1000,'Ali','Ahmadi',19)

t1=Teacher(100,'Reza','Akbari',20)

s1.show()

t1.show()

Student ...

Person ...

* کلاس های Enum : کلاس خاصی می باشد که برای زمانی است که تعداد زیادی از آن کلاس مورد نیاز نیست مانند کلاس هفته که 7 نمونه از آن داریم. برای ساخت کلاس Enum ابتدا باید ماژول enum را در برنامه import کرد سپس کلاس مورد نظر را از کلاس Enum و از ماژول enum می سازیم. به عنوان مثال کلاس رنگ ها را می خواهیم داشته باشیم که 4 رنگ بیشتر ندارد.

import enum

class Color(enum.Enum):

    Red = 1

    Blue = 2

    Green = 3

    Yellow = 4

print(Color.Yellow)

print(Color.Yellow.value)

print(Color.Yellow.name)

Color.Yellow

4

Yellow

فصل نهم

ماژول ها (Modules) در زبان برنامه نویسی پایتون

* ساخت و فراخوانی یک ماژول در زبان پایتون : زمانیکه یک برنامه می نویسیم که مجموعه ای از توابع ، کلاس و اشیا می باشد ممکن است در یک پروژه ای دیگر از همین توابع و کلاس و اشیا استفاده شود ، می توان آن مجموعه را تحت عنوان یک ماژول ساخت و با دستور import آن را در پروژه دیگر فرخوانی کرد. به عنوان مثال ما یک ماژول می سازیم که چهار عمل اصلی را انجام می دهد و سپس آن را در پروژه دیگر فرخوانی می کنیم .

myModule.py

------------------

def sum(a,b):

    return a+b

def sub(a,b):

    return a-b

def mul(a,b):

    return a\*b

def div(a,b):

    return a/b

p1.py

--------------------

import myModule

print(myModule.sum(10,20))

30

* تعریف کلاس ها ، اشیا و توابع در ماژول ها : همچنین می توان از کلاس و اشیا یک ماژول نیز استفاده نمود.

calassModule.py

--------------------------

class Person:

    def \_\_init\_\_(self,name,age):

        self.name = name

        self.age = age

    def \_\_str\_\_(self):

        return self.name+" "+str(self.age)

#--------------------------------------

p1=Person('mojtaba',37)

import classModule

p2=classModule.Person("Ali",32)

print(classModule.p1)

print(p2)

mojtaba 37

Ali 32

* تغییر نام ماژول به کمک as : با توجه به اینکه نام ماژول ها می بایست مرتبط باشد زمانی ممکن است نام آن بزرگ باشد لذا می توان هنگام import کرد از as استفاده کرد و یک نام مستعار به آن داد.

import myModule as m

print(m.mul(10,20))

* فراخوانی بخشی از ماژول به کمک دستور from : زمانی ممکن است ماژول شامل توابع و کلاس ها و اشیا زیادی باشد و مثلا فقط یک یا چند تا تابع از آن ماژول نیاز داریم ، پس می توانیم از کلمه from استفاده کرده و تنها همان توابع مورد نیاز را در پروژه استفاده کرد.

from myModule import mul,div

print(mul(20,10))

print(div(20,10))

200

2.0

* معرفی برخی از ماژول های داخلی پایتون ( Built-in Module )
* math module
* random module
* datetime module

فصل دهم

آموزش کار با انواع فایل ها در زبان برنامه نویسی پایتون : یکی از مباحثی که حتما تمام برنامه نویسان به آن نیاز دارند دسترسی به فایل های درون سیستم می باشد. برای این کار در تمام زبانهای برنامه نویسی یکسری دستورات ، یکسری پکیج ها و ماژول ها آماده شده است که می توان از آنها استفاده نمود. نکته مهم آن است که برنامه نویس بداند میخواهد به چه نوع فایلی دسترسی داشته باشد و قرار است چه نوع کاری آن فایل انجام دهد و نکته بعدی این است که قرار است چه کسی این خدمت را به آن بدهد. زیرا فایل ها تحت کنترل سیستم عامل ها هستند. پایتون به طور کلی فایل ها را به دو دسته می شناسد :

* معرفی انواع فایل ها
* فایل های متنی : همان فایل های تکستی هستند که درون آنها رشته ها و متن ها وجود دارند مانند فایل های که با notepad یا world ساخته می شوند.
* فایل های باینری : فایل هایی هستند که زمانیکه با برنامه های text editor باز می شوند متن های ناخوانا درون آن می باشد. مانند فایل هایی با پسوند .dll یا .bat یا .jar و غیره
* معرفی انواع روش های باز کردن فایل ها در زبان پایتون
* باز کردن فایل برای خواندن ( read ) : در این حالت می بایست فایل از قبل وجود داشته باشد تا فایل خوانده شود.

file1=open("C:/Files/f1.txt","r")

str1=file1.read()

print(str1)

file1.close()

نکته : می توان فایل ها به جای اینکه با متد read یخوانیم که فقط یک رشته برمی گرداند ، با متد readlines خواند و لیستی از رشته ها به ما نشان می دهد یعنی هر line از فایل را به عنوان یک عنصر لیست نشان می دهد

file1=open("C:/Files/f1.txt","r")

lines=file1.readlines()

print(lines)

file1.close()

['Mojtaba Shahroodi\n', 'False']

file1=open("C:/Files/f1.txt","r")

lines=file1.readlines()

for line in lines:

    print(line,end="")

file1.close()

Mojtaba Shahroodi

False

* باز کردن فایل برای نوشتن ( write ) : در این حالت حتی اگر فایل وجود نداشته باشد ایجاد می شود و توی آن نوشته می شود و اگر مجدد همان فایل با این حالت باز شود ، محتوای آن پاک و مجدد روی آن می نویسد.

file1=open("C:/Files/f1.txt","w")

file1.write("Mojtaba Shahroodi")

file1.close()

* باز کردن فایل برای اضافه کردن به انتهای فایل ( append ) : در این حالت به انتهای متن داخل فایل نوشته اضافه می شود.

file1=open("C:/Files/f1.txt","a")

file1.write("\n")

file1.write("False")

file1.close()

* باز کردن فایل برای نوشته با x : هر زمان یک فایلی را که مطمئن باشیم که از قبل وجود ندارد و بخواهیم آن را در لحظه اول کار بسازیم از متد x استفاده می کنیم پس این متدی است که باید فایل از قبل وجود نداشته باشد تا فایل را به صورت خالی ایجاد کند.

file1=open("C:/Files/f2.txt","x")

file1.close

* انواع روش های انجام عملیات خواندن از فایل ها : که در بالا توضیح داده شد.
* روش نوشتن اطلاعات در فایل ها : که در بالا توضیح داده شد.
* چک کردن وجود یا عدم وجود فایل: برای این کار می توان از ماژول os استفاده کرد ( معمولا هر زمان که بخواهیم از سیستم عامل استفاده کنیم یا یک درخواست به سیستم عامل دهیم احتمالا از این ماژول می توان استفاده کرد.) این ماژول یک کلاس به نام path دارد که درون آن تابعی به نام exists() وجود دارد که برای چک کردن وجود یا عدم وجود فایل به کمک می کند :

import os

if os.path.exists("C:/Files/f3.txt") :

    print("File is exists")

else:

    file1=open("C:/Files/f3.txt","x")

    file1.write("Ali Reza")

    file1.close()

نکته : تا الان در مثال های فوق با فایل هایی کار کردیم که خارج از پروژه ما و در شاخه c: بوده است ، اگر این پروژه را در کامپیوتر دیگر اجرا کنیم احتمال دارد اجرا نشود زیرا ممکن است آدرس ما در آن کامپیوتر وجود نداشته باشد پس برای آنکه این مشکل به وجد نیاید به صورت نسبی آدرس دهی می کنیم یعنی در پوشه ای که پروژه ما وجود دارد پوشه خود را می سازیم.

file1=open("files/f2.txt","w")

file1.write("Mehdi Abbasi")

file1.write("\n")

file1.write("False")

file1.close()

* حذف فایل : برای حذف فایل نیز می توان از ماژول os تابع remove استفاده نمود:

import os

os.remove("files/f2.txt")