آموزش زبان برنامه نویسی پایتون
بخش دوم

مرجع اصلی

http://w3schools.com

محمدجواد فدائی‌اسلام

۴ مهر ۱۳۹۹

باسمه تعالی

فهرست مطالب

۱- ورودی کاربر

تابع input

با این دستور پایتون منتظر می‌ماند تا ورودی از کاربر دریافت نماید و آن را در متغیر قرار دهد.

```
username = input("Enter username:")
print("Username is: " + username)
```

۲- تابع

با استفاده از کلمه کلیدی def، تابع ساخته می‌شود. در مثال زیر یک تابع ایجاد شده است.

```
def my_function():
  print("Hello from a function")
```

فراخوانی تابع با نوشتن نام آن به همراه پرانتز انجام می‌شود. تابع در هر جایی می‌تواند نوشته شود.

```
my_function()
```

تابع زیر دارای پارامتری به نام fname است.

```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

تابعی با دو پارامتر:

```
def my_function(fname, lname):
  print(fname + " " + lname)
my_function("Emil", "Refsnes")
```

فراخوانی تابعی با آرگومانی از جنس لیست:

```
def my_function(food):
  for x in food:
    print(x)
fruits = ["apple", "banana", "cherry"]
my_function(fruits)
```

- **پارامتری با مقدار پیش‌فرض**

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

- **ارسال آرگومان به صورت کلمه کلیدی**

ارسال آرگومان به صورت key=value
در این حالت ترتیب آرگومان‌ها مهم نیست.

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

- **تعداد آرگومان دلخواه**

اگر تعداد آرگومان معین نباشد، در زمان تعریف آرگومان یک ستاره پیش از نام آرگومان درج شود.

```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

- **تعداد آرگومان به صورت کلمه کلیدی دلخواه**

در زمان تعریف تایع دو ستاره پیش از نام آن درج شود.

```
def my_function(**kid):
  print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")
```

- **مقدار بازگشتی**

برای بازگشت مقدار، از کلمه کلیدی return استفاده می‌شود.

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

- **مثال تابع بازگشتی**

```
def tri_recursion(k):
 if(k > 0):
  result = k + tri_recursion(k - 1)
  print(result)
 else:
  result = 0
 return result

print("\n\nRecursion Example Results")
tri_recursion(6)
```

۳- تابع لاندا

یک تابع بی‌نام است که آرگومان‌های متعددی می‌گیرد. اما همیشه یک عبارت را محاسبه می‌کند.
ساختار آن به شکل زیر است.

```
lambda arguments : expression
```

در زیر مثال‌هایی مشاهده می‌شود که تابع لاندا به ترتیب یک و سه آرگومان دارد.

```
x = lambda a : a + 10
print(x(5))

x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

زمانی قدرت تابع لاندا مشخص می‌شود که در تابع دیگر تعریف شود.

```
def myfunc(n):
 return lambda a : a * n
```

دقت شود که برنامه بالا مقدار برنمی‌گرداند بلکه خروجی آن یک تابع است.

```
def myfunc(n):
 return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)

print(mydoubler(11))
print(mytripler(11))
```

برنامه بالا یک‌بار ۱۱ را در دو ضرب می‌کند و باردیگر در ۳.

۴- اسکوپ[۱] یا محدوده کاری متغیر

۱ - scope

متغیر در ناحیه‌ای که تعریف شده است شناخته می‌شود؛ به این موضوع اسکوپ گویند.
متغیری که در درون تابع تعریف می‌شود، تنها در همان تابع و توابع تعریف شده در درون آن معتبر است.

```
def myfunc():
 x = 300
 def myinnerfunc():
  print(x)
 myinnerfunc()

myfunc()
```

- **متغیر سراسری**[۲]

متغیری که در بدنه پایتون تعریف می‌شود، سراسری است. اگر در تابعی بخواهیم متغیر سراسری تعریف کنیم باید شناسه global را پیش از آن قرار دهیم.

```
def myfunc():
  global x
  x = 300
myfunc()
print(x)
```

- **متغیر هم‌نام**

اگر متغیر سراسری با متغیر محلی هم‌نام باشد متغیر محلی در محدوده تابع عمل می‌کند و متغیر سراسری بدون تغییر می‌ماند. به عبارتی به صورت دو متغیر جدا از هم هستند.

```
x = 300
def myfunc():
 x = 200
 print(x)

myfunc()
print(x)
```

اگر قرار است متغیر عمومی در درون تابع تغییر کند، باید به صورت زیر عمل شود.

```
x = 300
def myfunc():
 global x
 x = 200
myfunc()
print(x)
```

۲ - global

۵- ماژول[۳] یا کتابخانه

ماژول یک فایل است که در آن چند تابع وجود دارد و شما آن را به برنامه خود اضافه می‌نمایید.
برای ساخت یک ماژول باید توابع مورد نظر خود را در یک فایل با پسوند py. قرار داد.

مثال

تابع زیر در فایلی به نام mymodule.py ذخیره شده تا ماژولی به همین نام ایجاد شود.

```
def greeting(name):
  print("Hello, " + name)
```

- **به کارگیری ماژول‌ها**

با دستور import می‌توان یک ماژول را به برنامه ضمیمه نمود. برای استفاده از یک تابع که در ماژول تعریف شده باید از ساختار زیر بهره گرفت:

module_name.function_name.

مثال

```
import mymodule
mymodule.greeting("Jonathan")
```

اگر در ماژول متغیری تعریف شده باشد مانند به کارگیری توابع می‌توان از آن بهره گرفت.

مثال

متغیر دیکشنری زیر به ماژول mymodule.py اضافه شده است.

```
person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

به صورت زیر از این دیکشنری استفاده شده است.

```
import mymodule

a = mymodule.person1["age"]
print(a)
```

- **استفاده از ماژول با نام جدید**

این کار با استفاده از دستور as میسر است.

---

۳ - module

```
import mymodule as mx

a = mx.person1["age"]
print(a)
```

- **تابع dir()**

با این تابع می‌توان نام تمام توابع درون یک ماژول را لیست نمود. دستورات زیر نام تمام توابع داخل ماژول platform را چاپ می‌نماید.

```
import platform

x = dir(platform)
print(x)
```

- **فراخوانی قسمتی از یک ماژول با دستور from**

```
def greeting(name):
  print("Hello, " + name)
person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

```
from mymodule import person1
print (person1["age"])
```

نکته: همان‌طور که در مثال دیده می‌شود زمانی که از from استفاده می‌شود. نباید نام ماژول را ذکر کرد. کاربرد زیر نادرست است.

```
mymodule.person1["age"]
```

۶- تاریخ

در پایتون نوع داده‌ای به نام تاریخ نداریم، اما ماژول datetime برای این کار در نظر گرفته شده است.

مثال زیر دستوراتی برای نمایش زمان جاری را نمایش می‌دهد.

```
import datetime
x = datetime.datetime.now()
print(x)
```

```
print(x.year)
```

خروجی دستورات بالا به صورت زیر است:

```
2020-09-24 10:15:42.149172

2020
```

توابع زیادی برای کار با زمان در این ماژول وجود دارد.

۷- ریاضی

- **توابع ریاضی**

تعدادی تابع در پایتون برای ریاضی وجود دارد

```
x = min(5, 10, 25)
y = max(5, 10, 25)

a = [10, 5, 6, 19, 7]

z = min(a)

x = abs(-7.25)

x = pow(4, 3) # x = 64
```

- **ماژول ریاضی**

علاوه بر توابع که تعدادی از آنها در بالا ذکر شده است، ماژول ریاضی هم در پایتون وجود دارد.

```
import math

x = math.sqrt(64)

x = math.ceil(1.4)
y = math.floor(1.4)
print(x) # returns 2
print(y) # returns 1

x = math.pi
```

۸- بسته

تمام فایل‌های مورد نیاز برای یک ماژول، بسته[۴]نام دارد. بسته‌های متعددی برای پایتون ایجاد شده است که کاربردهای گوناگونی دارند. PIP خود یک بسته است که برای مدیریت بسته‌های پایتون طراحی

[۴] - package

شده است. اگر بسته‌ای نصب نباشد می‌توان با اجرای دستور زیر در خط فرمان آن را نصب نمود. به طور مثال دستور زیر که در خط فرمان اجرا شده است، بسته camelcase را نصب می‌نماید.

```
>pip install camelcase
```

دستور زیر ویرایش بسته pip را نشان می‌دهد.

```
>pip --version
```

دستور زیر بسته نصب شده camelcase را حذف می‌کند.

```
>pip uninstall camelcase
```

دستور زیر بسته‌های نصب شده در سیستم را لیست می‌کند.

```
>pip list
```

پس از نصب بسته می‌توان آن را با استفاده از دستور import مانند یک ماژول در برنامه به کار برد.

۹- try ... except

وجود خطا در برنامه باعث می‌شود که برنامه خاتمه یابد اما با دستورات زیر می‌توان آن را مدیریت نمود.

```
try:
  print(x)
except:
  print("An exception occurred")
```

در برنامه بالا متغیر x تعریف نشده است اگر در غالب try استفاده نشود باعث ایجاد خطا خواهد شد. اما در برنامه بالا یک پیام نمایش داده خواهد شد.

- **else**

```
try:
  print("Hello")
except:
  print("Something went wrong")
else:
  print("Nothing went wrong")
```

در کد بالا درصورت نبود مشکل قسمت else اجرا می‌شود.

- **finally**

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

در کد بالا درهرصورت (بروز یا عدم بروز خطا) قسمت finally اجرا خواهد شد و کد ادامه خواهد یافت.

۱۰- فایل

- **باز نمودن فایل**

فایل با دستور open به چهارحالت باز می‌شود. علاوه برآن می‌توان فایل را به صورت متنی (t) یا باینری (b) باز نمود.

r	read	برای خواندن، در صورت نبود فایل خطا می‌دهد.
a	append	اضافه نمودن داده به آخر فایل، در صورت نبود فایل ایجاد می‌شود.
x	create	ایجاد فایل، در صورت وجود فایل خطا می‌دهد.
w	write	نوشتن فایل، در صورت نبود فایل ایجاد می‌شود

```
f = open("demofile.txt", "rt")
```

فایل demofile را به صورت خواندنی و به حالت متنی باز می‌کند. در صورت معین ننمودن نوع بازکردن فایل به صورت پیش‌فرض متنی، خواندنی باز می‌شود.

- **خواندن فایل**

خواندن تمام فایل به صورت زیر است.

```
f = open("D:\\myfiles\welcome.txt", "r")
print(f.read())
```

در دستور زیر ۵ کاراکتر اول فایل خوانده می‌شود.

```
f = open("demofile.txt", "r")
print(f.read(5))
```

دستور زیر خواندن خط به خط است که می‌توان در حلقه از آن کمک گرفت.

```
f = open("demofile.txt", "r")
print(f.readline())
print(f.readline())
```

پس از کار با فایل، آن را باید بست.

```
f.close()
```

- **نوشتن/ایجاد در فایل**

در دستور زیر متنی به انتهای فایل اضافه می‌شود.

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()
```

- **حذف فایل و دایرکتوری**

دستور زیر فایل demofile را حذف می‌کند.

```
import os
os.remove("demofile.txt")
```

اگر فایل وجود نداشته باشد خطا می‌گیرد، برای جلوگیری از آن به صورت زیر می‌توان نوشت.

```
import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

برای حذف داریکتوری به صورت زیر می‌توان عمل کرد. البته باید دایرکتوری خالی باشد.

```
import os
os.rmdir("myfolder")
```

۱۱- تمرین

۱- تابعی بنویسید که nامین عدد از رشته فیبوناچی را محاسبه نماید (به دو صورت مستقیم و بازگشتی)

۲- برررسی نمایید که ورودی‌های توابع مثلثاتی به درجه است یا رادیان؟ برای تمرین در یک برنامه از آنها استفاده شود (ورودی به درجه و گرادیان دریافت کند و حاصل را نمایش دهد.)

۳- جمع هزار عدد اول را محاسبه نماید.

۴- فایل متنی باز نماید. کلمات آن را در داخل دیکشنری ذخیره کند. روبروی هر مدخل در دیکشنری تعداد تکرار آن است.

۵- مقدار زمان سپری شده برای اجرای الگوریتم فیبوناچی به صورت بازگشتی را به ازاری ورودی دلخواه n محاسبه نماید.

*۶- فایلی متنی دریافت کند و ان را با استفاده از روش هافمن کد نماید. درخت جستجو را در داخل یک دیکشنری ذخیره نماید.