



Object Oriented Programming using Python (I)

Lecture(3)

Python programming language(function)

Prepared by: **Dr. Rula Amjed Hamid**
University of Information Technology and Communications
College of Business Informatics

Function

- A **function** is a unit of reusable code
- Code is made more reusable by packaging it in functions.
- we will see how to write our own reusable functions and examine some of functions available in the Python standard library.
- Python provides a collection of standard code stored in libraries called **modules**.
- Programmers can use parts of this library code within their own code to build their programs.

Built-in Functions

Python has several functions that are readily available for use. These functions are called built-in functions. In this reference page, you will find all the built-in functions in Python.

example

This program use **sqrt** function to find square root of number and **math** library that contain sqrt function

```
from math import sqrt

# Get value from the user
num = eval(input("Enter number: "))

# Compute the square root
root = sqrt(num)

# Report result
print("Square root of", num, "=", root)
```

Function

```
from math import sqrt
```

- makes the sqrt function available for use in the program.
- Math is **Module** a collection of Python code that can be used in other programs .
- math module has many other mathematical functions. These include logarithmic, hyperbolic, and other mathematical functions. When calling a function, the function's name is followed by parentheses that contain the information to pass to the function so it can perform its task.

```
sqrt(num)
```

Math functions module

| mathfunctions Module | |
|----------------------|--|
| sqrt | Computes the square root of a number: $\text{sqrt}(x) = \sqrt{x}$ |
| exp | Computes e raised a power: $\text{exp}(x) = e^x$ |
| log | Computes the natural logarithm of a number: $\text{log}(x) = \log_e x = \ln x$ |
| log10 | Computes the common logarithm of a number: $\text{log}(x) = \log_{10} x$ |
| cos | Computes the cosine of a value specified in radians: $\text{cos}(x) = \cos x$; other trigonometric functions include sine, tangent, arc cosine, arc sine, arc tangent, hyperbolic cosine, hyperbolic sine, and hyperbolic tangent |
| pow | Raises one number to a power of another: $\text{pow}(x,y) = x^y$ |
| degrees | Converts a value in radians to degrees: $\text{degrees}(x) = \frac{\pi}{180}x$ |
| radians | Converts a value in degrees to radians: $\text{radians}(x) = \frac{180}{\pi}x$ |
| fabs | Computes the absolute value of a number: $\text{fabs}(x) = x $ |

- The math package also defines the values(pi)and (e).

examples

```
# This program shows the various ways the
# sqrt function can be used.

from math import sqrt

x = 16
# Pass a literal value and display the result
print(sqrt(16.0))
# Pass a variable and display the result
print(sqrt(x))
# Pass an expression
print(sqrt(2 * x - 5))
# Assign result to variable
y = sqrt(x)
print(y)
# Use result in an expression
y = 2 * sqrt(x + 16) - 4
print(y)
# Use result as argument to a function call
y = sqrt(sqrt(256.0))
print(y)
print(sqrt(int('45')))
```

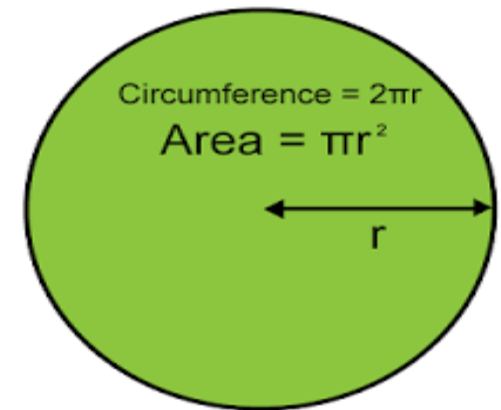
■ Also We can use math functions as following:

`y = math.sqrt(x)`

examples

This program to find circumference and area of circle

```
from math import *  
  
r=eval(input("enter raduis of circle  
x=2*r*pi  
y=pow(r,2)*pi  
print("circumference of a circle",x)  
  
print("area of a circle",y)  
  
print(pi)
```



Random functions

Some applications require behavior that appears random. Random numbers are useful particularly in games and simulation.

randomfunctions Module

random

Returns a pseudorandom floating-point number x in the range $0 \leq x < 1$

randrange

Returns a pseudorandom integer value within a specified range.

```
from random import randrange

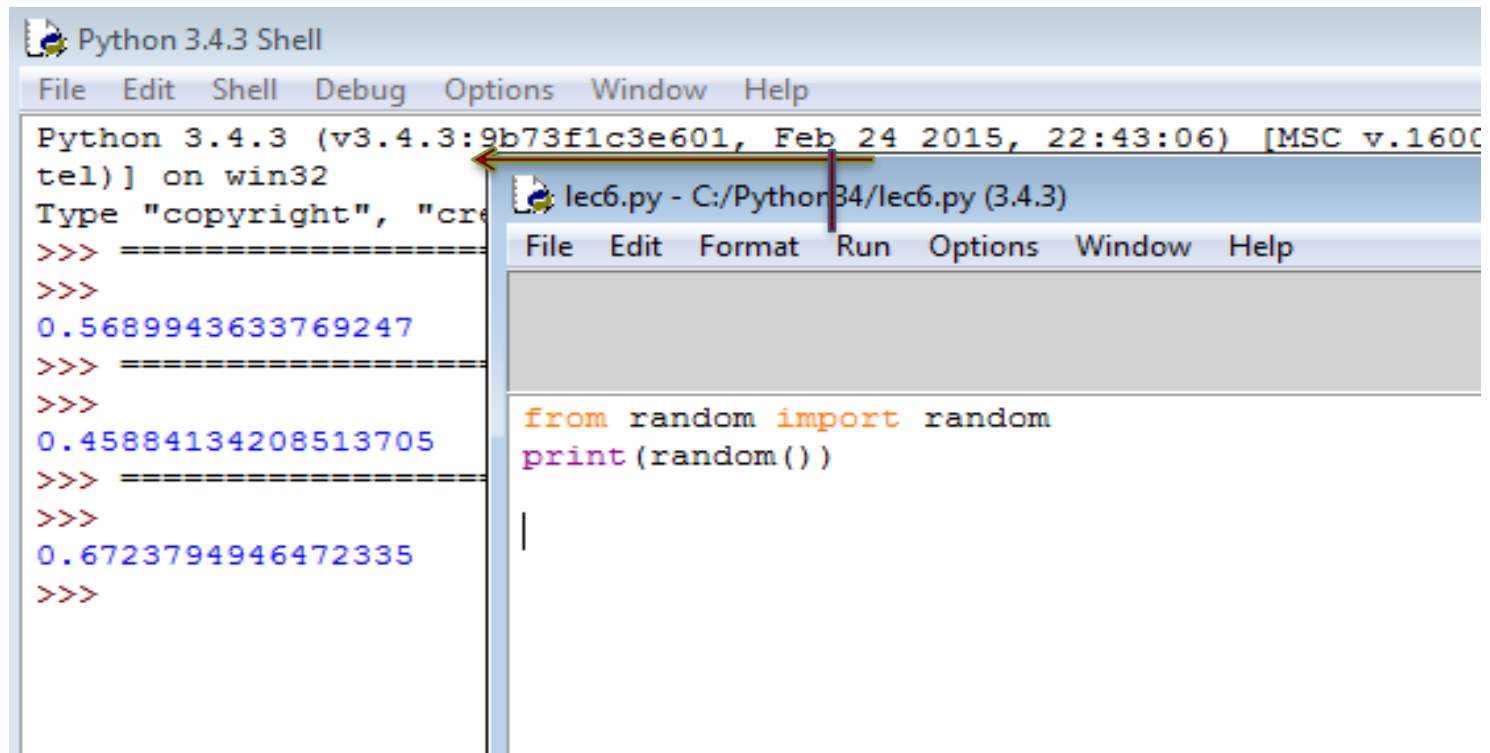
for i in range(0, 20):
    print(randrange(1, 100), end=' ')
print()
```

run

```
>>>
96 6 91 42 39 18 32 41 47 86 15 94 27 13 54 32 27 71 36 94
>>> |
```


Random function

Some functions do not accept any parameters; for example, the function to generate a pseudorandom floating-point number, `random` function requires no arguments:



The image shows two overlapping windows from a Python 3.4.3 environment. The background window is the 'Python 3.4.3 Shell' with a menu bar (File, Edit, Shell, Debug, Options, Window, Help). It displays the Python version and build information, followed by a copyright notice. The prompt is `>>>`. Three calls to the `random` function are shown, each preceded by a separator line of equals signs. The outputs are `0.5689943633769247`, `0.45884134208513705`, and `0.6723794946472335`. The foreground window is a script editor titled 'lec6.py - C:/Python34/lec6.py (3.4.3)' with a menu bar (File, Edit, Format, Run, Options, Window, Help). It contains the following code:

```
from random import random
print(random())
|
```

Example Max () function

- The max() function returns the largest of the input values.

```
# find largest item in the string
print(max("abcDEF"))

# find largest item in the list
print(max([2, 1, 4, 3]))
```

- <https://thepythonguru.com/python-builtin-functions/>

Using Local Variables

Scope of Variables

All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable. The scope of a variable determines the portion of the program where you can access a particular identifier. There are two basic scopes of variables in Python –

Global variables

Local variables

Global vs. Local variables

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.

This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions. When you call a function, the variables declared inside it are brought into scope.

Using Local Variables

```
def fun1(x):  
    print("x is " ,x)  
    x=2  
    print("change local x to ",x)|  
x=50  
fun1(x)  
print("x is still",x)
```

X in main program is
different than x that define
within a function

```
///  
>>>  
x is 50  
change local x to 2  
x is still 50  
>>>
```

Using Local Variables

```
def increment(x):  
    print("Beginning execution of increment, x =", x)  
    x += 1 # Increment x  
    print("Ending execution of increment, x =", x)  
  
def main():  
    x = 5  
    print("Before increment, x =", x)  
    increment(x)  
    print("After increment, x =", x)  
  
main()
```

The result

```
>>>  
Before increment, x = 5  
Beginning execution of increment, x = 5  
Ending execution of increment, x = 6  
After increment, x = 5  
>>> |
```

Using the global statement

Is used to declare a global variable when we give the value of the variable in the function of this change it appears in the main program

```
def fun1():  
    global x  
    print("x is " ,x)  
    x=2  
    print("change global x to ",x)  
x=50  
fun1()  
print("value of x is ",x)  
|
```



```
'''  
x is 50  
change global x to 2  
value of x is 2  
>>>
```