Functions

Function calls

- In the context of programming, a function is a named sequence of statements that performs a computation.
- When you define a function, you specify the name and the sequence of statements. Later, you can "**call**" the function by name.
- Examples of a function call:

```
print(123)
print("Python")
type('abc')
```

Function calls

- To use a function that is defined in a module, a program must *import the module*, using keyword import. After, the program can invoke functions in that module, using the call:

  moduleName.functionName()

- Example:

```
import math
print(math.sqrt(25))
```

Type conversion functions

```
int('32')
int('hello')    # error
int(3.9999)     # no round off
int(-2.645)

float(5)
float('3.14159')

str(5)
str(3.14159)
```

Function Definitions

- The format of a function definition is:

```
def function-name( parameter-list ):
        statements
```

- Example 1:

```
def print_lyrics():
    print("I'm a student, and I'm okay.")
    print("I sleep all night and I work all day.")

print_lyrics()                  # call function
```

Function Definitions

- Example 2:

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

for x in range( 1, 11 ):
  print(square(x))
```

Pass by Reference or pass by value

- In Python every variable name is a reference. When we pass a variable to a function, a new reference to the object is created

```
def myFun(x):
   x[0] = 20

lst = [10, 11, 12, 13, 14, 15]
myFun(lst);
print(lst)
```

Output:

```
[20, 11, 12, 13, 14, 15]
```

Pass by Reference or pass by value

- When we pass a reference and change the received reference to something else, the connection between passed and received parameter is broken.

```
def myFun(x):
   x = 20

x = 10
myFun(x);
print(x)
```

Output:

```
10
```

Pass by Reference or pass by value

```
def swap(x, y):
    temp = x;
    x = y;
    y = temp;

x = 2
y = 3
swap(x, y)
print(x)
print(y)
```

Default arguments

- A default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument.

```
def myFun(x, y=50):
    print("x: ", x)
    print("y: ", y)

myFun(10)
```

Output:

```
x:  10
y:  50
```

Keyword arguments

- The programmer can specify that a function receives one or more *keyword arguments*. The function definition assigns a default value to each keyword.
- A function may use a default value for a keyword or a function call may assign a new value to the keyword using the format **keyword = value**.
- When using keyword arguments, the position of arguments in the function call is not required to match the position of the corresponding parameters in the function definition.

Example:

```
def F(x,y):
  print(x,y)
F(x = 5, y = 2)
F(y = 2, x = 5)
```

```
Output:
5 2
5 2
```

Variable length arguments

- .
```
def F(*argv):
    for arg in argv:
        print (arg)
F(5, 2, ‘ab’, 12)
```

Lambda function

- Lambda functions are created without using **def** keyword and without a function name
- Syntax: `lambda` *argument_list*:*expression*
- Example:

```
binhphuong = lambda x:x*x
print(binhphuong(2.5))
```

Generator

- A generator is something that you can iterate over but whose values are produced only as needed.

```
def simpleGeneratorFun():
    yield 1
    yield 2
    yield 3


for value in simpleGeneratorFun():
    print(value)
```

Generator

```
def nextSquare():
    i = 1;
    while True:
        yield i*i
        i += 1


for num in nextSquare():
    if num > 100:
         break
    print(num)
```

Object-Oriented Programming

- Like many languages, Python allows you to define classes that encapsulate data and the functions that operate on them.
- Imagine we didn’t have the built-in Python set. Then we might want to create our own Set class.

Object-Oriented Programming

```
class Set:
# these are the member functions add, remove, contains
# every one takes a first parameter "self" (another convention)
# that refers to the particular Set object being used
def __init__(self, values=None):
    """This is the constructor.
    It gets called when you create a new Set.
    You would use it like
    s1 = Set() # empty set
    s2 = Set([1,2,2,3]) # initialize with values"""
    # each instance of Set has its own dict property which is what we'll use to track memberships
self.dict = {}
if values is not None:
  for value in values:
    self.add(value)
```

Object-Oriented Programming

```
def __repr__(self):
    return "Set: " + str(self.dict.keys())

def add(self, value):
    self.dict[value] = True

def contains(self, value):
    return value in self.dict

def remove(self, value):
    del self.dict[value]
```