Scoping

- For variables, Python has function scope, module scope, and global scope.
- Name resolution is done using local, enclosing, global, built-in chain.
- The the global keyword to change global variables inside a function.
- Python supports nested functions and closures.

Scoping

```
def print_num():
    print(num)

def change_num():
    global num
    num = 18

num = 5
print_num()
change_num()
print_num()
```

Function Arguments

- Functions can have default arguments.
- Only immutable type should be used as default arguments.
- Variables with default arguments must come after the variables without default arguments.
- When the function is called, the arguments can be passed in a usual way one after another (normal arguments) or by assigning their names to values inside the call (keyword arguments).

Function Arguments

```
def f(arg1, arg2, arg3=5):
    pass

f(6, arg2=8)
```

Function Arguments

- The keyword arguments must come after the normal arguments.
- When calling a function, a list or a tuple can be expanded to normal function arguments using *args syntax and a dictionary can be expanded to a keyword arguments using **kwargs syntax.
- Similarly, *args and **kwargs syntax can be used inside a function declaration to declare a list of normal arguments and a dictionary of keyword arguments respectively.

Function Arguments

```
def f(*args, **kwargs):
    assert args == (6, 8)
    assert kwargs == {‘arg2’: 8}

f(6, 8, arg2=8)
```

Function Arguments

```
def f(n, m, animal):
    print(n + m)
    print(animal)

f(*[6, 8], **{‘animal’: ‘whale’})
```

Functions

- Functions are first-class citizens.
  - Functions can be assigned to variables.
  - Functions can be passed to other functions.
- Lambda expressions can be defined with lambda keyword.

Functions

```
def plus_one(func):
    return func() + 1

def five():
    return 5

plus_two = lambda x: x+2

print(plus_one(five)
print(plus_two(five()))
```

Functions

- Python supports nested functions and closures.
  - Functions can be defined inside other functions.
  - Inner functions can access the variables of the nesting functions even after the nesting functions are not executed.
- Decorator is a function that get an argument function and returns an inner function which wraps the argument function.
- Python has special syntax for using decorators.

Functions

```
def plus_one(func):
    def inner(num):
        return func(num) + 1
    return inner

def square(num):
    return num * num

square_plus_one = plus_one(square)

print(square_plus_one(2))
```

spam.py

```
from email_client import EmailClient

def send_spam(to):
    text = (“The best washing machines in town!\n\n”
            “Only now you can buy a new washing machine “
            “for the cheapest price ever. For more info ”
            “go to www.best-washing-machines.com”)
from = “master@best-washing-machines.com”
subject = “Hi”

client = EmailClient(‘smtp.server.com’)
client.send(from, to, subject, text)
```

spam.py

```
def send_spam(to):
    ...
if __name__ == '__main__':
    recipients = ['recipient1@mail.com', 'recipient2@mail.com']
    for recipient in recipients:
        send_spam(recipient)
```

spam.py

```
from multiprocessing import Pool

def send_spam(to):
    ...

if __name__ == '__main__':
    recipients = ['recipient1@mail.com', 'recipient2@mail.com']
    pool = Pool()
    pool.map(send_spam, recipients)
```

Parallel Grep

Data Model

- Objects are Python's abstraction for data.
- All data in a Python program is represented by objects or by relations between objects.
- Every object has an identity, a type and a value.
- Objects whose value can change are said to be mutable; objects whose value is unchangeable once they are created are called immutable.

Data Model

- Objects identity can be received using the id() built in function.
- Objects can be compared for identity using is keyword.
- For instance (1, 2) is (1, 2) == False.
- However, if v == None then (v is None == True).

Classes

```
class Name:
    <statement-1>
    .
    .
    .
    <statement-N>
```

Classes

- Classes (as well as modules or even local variables of functions) can be seen as a kind of a namespace — a mapping from names to objects.
- The statements in class definition define that mapping.
- The lookup for a name always begins with the most internal namespace to the most external.
- We use namespace.attribute syntax to access the internal names of a namespace.

Inheritance

```
class DerivedClassName(BaseClassName):
    <statement-1>
    .
    .
    .
    <statement-N>
```

Inheritance

- Use the built-in function isinstance() to check an instance's type: isinstance(obj, int) will be True only if obj.__class__ is int or some class derived from int.

- Use issubclass() to check class inheritance: issubclass(bool, int) is True since bool is a subclass of int. However, issubclass(float, int) is False since float is not a subclass of int.

Inheritance

- Method references are resolved as follows: the corresponding class attribute is searched, descending down the chain of base classes if necessary, and the method reference is valid if this yields a function object.

- An overriding method in a derived class may call the base class method directly BaseClassName.methodname(self, arguments) or using the built-in function super().methodname(self, arguments)

- With single inheritance, built-in function super() returns a proxy object that delegates method calls to a parent class.

Inheritance

```
class DerivedClassName(Base1):
    def __init__(self):
        Base1.__init__(self)
        ...
```

Multiple Inheritance

```
class DerivedClassName(Base1, Base2):
    def __init__(self):
        Base1.__init__(self)
        Base2.__init__(self)
        ...
```

Special Method Names

- A class can implement certain operations that are invoked by special syntax (such as arithmetic operations or subscripting and slicing) by defining methods with special names.

- This is Python's approach to operator overloading, allowing classes to define their own behavior with respect to language operators.

Special Method Names

```
>> class AddTen:
>>     def __add__(self, n):
>>         return n + 10
>>
>> ten = AddTen()
>> ten + 5
15
```

Special Method Names

```
>> class Callable:
>>     def __call__(self, n):
>>         print(n)
>>
>> f = Callable()
>> f(5)
5
```

PyPI and PIP

- PyPI - the Python Package Index.
- As of December 2016, There are > 90K packages in PyPI.
- A package from a repository can be install using PIP utility.
- Some packages can also be installed from a (linux) os repository.

PyPI and PIP

- To install a package using pip in ubuntu type:
  - pip3 install package-name
- To install a package using pip in windows type:
  - pip install package-name

Flask Web Server

Flask Web Server #2