CS 38003 PYTHON PROGRAMMING

Ruby Tahboub

ARGUMENTS, VARIABLES, and CLASSES

DEFAULT ARGUMENTS (REVIEW)

- The arguments of a function may have default values.
- Functions can be called using parameter names e.g., `multiply (x=10, y=20)`

```
def multiply(x=10, y=20):
  return x * y

>>>print (multiply(10,5))
50

>>>print (multiply(3))
60

>>>print (multiply())
200

>>>print (multiply(x=5, y=3))
15

>>>print (multiply(y=3))
30
```


Slides adapted from UPenn CIS530 python tutorial

Running a Python Script from COMMAND LINE

- Write your script in a file
  - Pass the file name and (optional) arguments to python's interpreter.

fact.py

```
def factorial(n):
    result = 1
    for i in range(1, n+1):
        result = result * i
    return result

print (factorial(5))
```

```
$ python fact.py

120
```

Passing Command Line Arguments

- sys.argv contains the list of arguments passed to the script.

test_argv.py

```
import sys
for arg in sys.argv:
 print (arg)
```

```
$ python test_argv.py 123 abc
test_argv.py
123
abc
```

Passing Variable Number of Arguments

- `*arg` in function definition allows passing variable number of arguments.

```
def printArgs(first, *arg):
    print ("first argument =", first)
    for ar in arg:
        print (ar)

printArgs(10,1,2,3,4,5)
```

```
first argument = 10
1
2
3
4
5
```

Passing Variable Number of Key-Word-ed Arguments

- `**kwarg` in function definition allows passing variable number of key-word arguments.
- `kwarg` is a dictionary.

```
def printArgs(first, **kwarg):
    print (type(kwarg))
    print ("first argument =", first)
    for k, w in kwarg.items():
        print (k, "=",  w)

printArgs(10, a = 1, b = 2, c = 3, d = 4, e = 5)
```

```
<class 'dict'>
first argument = 10
a = 1
b = 2
c = 3
d = 4
e = 5
```

Built-in variable ___name___

- Recall, Python does not have a main function.
- Program execution starts at level 0 indentation.
- Just before program execution, Python sets the built-in variable __name__
  - If the source file is executed as the main program, the interpreter sets the __name__ variable to have a value "__main__"
  - If this file is being imported from another module, __name__ will be set to the module's name.

Built-in variable __name__

```
print ("testfile1 __name__ = %s" %__name__)
if __name__ == "__main__":
    print ('testfile1 is being run directly')
else:
    print ('testfile1 is imported')
```

```
import testfile1

print ("testfile2 __name__ = %s" %__name__)
if __name__ == "__main__":
  print ('testfile2 is being run directly')
else:
  print ('testfile2 is imported')
```

$python testfile1.py

testfile1 __name__ = __main__

testfile1 is being run directly

$python testfile2.py

testfile2 __name__ = testfile1

testfile1 is imported

testfile2 __name__ = __main__

testfile2 is being run directly

Built-in variable __name__

```
def main():
    print("Running the main function!")

if __name__ == "__main__":
    main()
```

$python test_main.py

Running the main function!

LOCAL and GLOBAL VARIABLES

LOCAL and GLOBAL VARIABLES

- Variables defined inside a code block are called local.
- Local variables are only visible within the block where they were defined.
- Global variables are defined outside all code blocks.
  - Global variables can be accessed anywhere in the program unless a local variable is defined then the local variable value is the one that can be accessed.
- global keyword is used to modify a global variable inside a function.

LOCAL and GLOBAL VARIABLES

```
s = "I enjoy programming"
def f():
    s = "Me too"
    print (s)

f()
print (s)
```

Me too

I enjoy programming

```
s = "I enjoy programming"
def f():
    global s
    print (s)
    s = "Me too"
    print (s)

f()
print (s)
```

I enjoy programming

Me too

Me too

CLASSES

CLASSES

- A 'class' is a user-defined data type.
- An instance of a class is called an object e.g., String is a class, ‘abc’ is an instance of the class String.
- A class contains “member data” (often called class attribute), and it also has associated things it can do, or “member functions”
- Instances are objects that are created which contain all of the member data and member functions from the class definition.

CLASSES

- Typically, a class consists of the following components:
  - Fields or member variables or attributes.
  - Methods or member functions.
  - Constructor (a special method) responsible for the creation of objects.

DEFINING A CLASS

- Define a method in a class by including function definitions within the scope of the class block.
- A constructor for a class is a method called __init__ defined within the class.
- There must be a special first argument self in all of method definitions which gets bound to the calling instance.
- `Self` is like this in Java.
- `Self` always refers to the current class instance

```
class Point:
    def __init__(self, xVal, yVal):
        self.x = xVal
        self.y = yVal

    def move_point(self, dx, dy):
        self.x = self.x + dx
        self.y = self.y + dy

    def print_point(self):
        print (self.x, self.y)


p = Point(3, -4)
p.print_point()
p.move_point(2,2)
p.print_point()
```

3 -4

5 -2

THE self KEYWORD

- All member functions (including constructor) should have the first parameter to be self.

```
def name(self, parameter, ..., parameter):
    statements
```

- Self represents the "implicit parameter“ – pointer to object itself.
- must access the object's fields through the self reference.

```
class Point:
    def move_point(self, dx, dy):
        self.x = self.x + dx
        self.y = self.y + dy
```

IMPLICIT and EXPLICIT self

- The self value can be an implicit or explicit parameter
  - object.method(parameters)
  - Class.method(object, parameters)

```
p = Point(3, -4)
p.move(1, 5) #self is implicit
Point.move(p, 1, 5) #self is explicit
```

OBJECTS ASSIGNMENT

```
>>>p1 = Point()
>>>p1.x = 7
>>>p1.y = -3
>>>p2 = p1 # p1 is assigned to p2
>>>p2.x = 10
>>>p2.print_point()
(10, -3)
>>>p1.print_point()
(10, -3)
```

THANK YOU!