Basic Python Programming

Yilin Zheng

Dept. of Computer Science and Engineering

June 28, 2017

Outline

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Conditional statement

- Conditional statement has **Boolean value** `True` or `False`

if, else and elif

- `if` and `elif` take an **conditional statement**
- Form: `if conditional statement:`
  `elif conditional statement:`
  `else:`

while

- Form: `while conditional statement:`

for

- `for` needs to **indicate a range**
- Use `in...range()` to indicate the range
- `range()` takes two number as one is the beginning, the second is the end(excluded)
- Form: `for conditional statement:`

break and continue

- `break` is used to **stop** and **escape from loop**
- Can be used both in `while` and `for` loop
- `continue` is to **skip the subsequent codes in current loop and directly execute next loop**

Function

- Function is **a block of codes with a name**
- Can be used in any place of programs by **calling the function name with parameters if needed**, which is called **function call**
- Some built-in functions we have ever used before like `len` and `range`

Define a function

- Use key word `def`
- Following the `def`, we need **a function name**
- Following the name, we add **parameters** which function may take
- Form:

```
def Function_Name(parameters):
    #Code blocks
    return #Better end with return regardless
                                of whether need to
                                return value or not
```

- If no `return`, function will return `None`

Example

- This function takes a parameter and conut the bits of 1 in its binary form

```
#!/usr/bin/python
# Filename: function.py
number = int(input("Enter an integer:"))
def count_Bit(number):
    n = 0
    while number:
        n += (number&1)
        number = number>>1
    return n
print("There are", count_Bit(number),"bits of 1
                        in integer", number)
```

Parameters

- Parameters, also called **formal parameters**, **take the value you pass to function** then used by functions
- Parameters are **specified in parentheses** and separated by **comma ,**
- When calling functions, we should pass value **in corresponding order**

```
#!/usr/bin/python # Filename: function.py
def printMax(a, b):
    if a > b:
        print(a, 'is maximum')
    else:
        print(b, 'is maximum')
    return
printMax(25, 36) # a = 25, b = 36, 25 and 36
                 are actual parameters
```

Default parameter value

- When some parameters are optional, we don't want to pass values for these parameters, then we should set default for these parameters
- Use = with **the default value**
- Example

```
#!/usr/bin/python
# Filename: function.py
def say(message, times = 1):
    print(message * times)
say('Hi') #default value is 1, so print once
say('Biu~', 3) #print 3 three times
```

- Default parameter value **cannot be modified**, and **can only be set in the last**

Key parameter

- When we want to pass values for some parameters but not all of them, we can use **parameter name** with = to set the value
- Example

```
#!/usr/bin/python # Filename: function.py
def print_abc(a, b=5, c=10):
    print('a is', a, 'and b is', b, 'and c is'
                                , c)
print_abc(2, 8) #a = 2, b = 8, c = default
                            value 10
print_abc(76, c=3.1415) #a = 76, b =  default
                            value 5, c = 3.1415
print_abc(c=100, a=4+10j) #a = 4+10j, b =
                            default value 5, c =
                            100
```

- It depends on **the name of parameter but not its position**

Local variable

- The variables defined within functions **has nothing to do with** the variables defined outside functions
- So they can have the same name with external variables
- Variable's name is locally valid, we called this **the scope of variables**

```
#!/usr/bin/python
# Filename: function.py
x = 50
def print_Local(x):
    print('x is', x) #x = 50
    x = 2
    print('Changed local x to', x) #x = 2
print_Local(x) #print 2
print('x is still', x) #print 50
```

global

- What if we want to change the external variables' value within funtions?
- Add `global` before the name

```
#!/usr/bin/python # Filename: function.py
x = 50
def print_global():
    global x
    print('x is', x) #print 50
    x = 2 #external x has been changed to 2
    print('Changed local x to', x)
print_global()
print('Value of x is', x)#print 2
```

- In this function, x has global scope so original value has been modified

return

- Use `return` to **indicate the end of function** or **return results**

```
#!/usr/bin/python
# Filename: function.py
def print_max(x, y):
    if x > y:
        return x#return value of x
    else:
        return y#return value of y
print(print_max(2, 3))
```

- If no `return`, functions will return `None` since functions have underlying `return None` in that case
- Also use `pass` to represent an **empty statement block**

DocStrings

- DocStrings can help your codes more comprehensible
- It has fixed position, is **the first expression within the function** and specified in ''' '''
- It is substantially a **multi-line strings**
- **First line** begins with **capital letter** and ends with **period** .
- **Second line** leaves **empty**
- **Third line and later lines** are used to **describe the detail**
- Use `.__doc__` (**double underscore**) to refer to DocStrings of a function, e.g. `function_name.__doc__`
- Use built-in function `help` is also fine, e.g. `help(function_name)`

Example 1

- Use .__doc__

```
#!/usr/bin/python
# Filename: DocStrings.py
def printMax(x, y):
    '''Print the maximum of two numbers.

    The two values must be integers.'''
    x = int(x) # convert to integers, if
                        possible
    y = int(y)
    if x > y:
        print(x, 'is maximum')
    else:
        print(y, 'is maximum')
printMax(10, 5)
print(printMax.__doc__)
```

Example 2

- Use `help`

```
help(printMax) # ==  print(printMax.__doc__)
```