Python Function

江琳

Philosophy

- **A function is a block of organized, reusable code that is used to perform a single, related action.**
- **They allow you to give a name to a block of statements and you can run that block using that name anywhere in your program and any number of times.**
- **We have already used many built-in functions such as the print(), len() and range().**

Defining a function

- **Syntax**

```
def functionname(parameters):
    function_suite
    return [expression]
```

Example

```
from datetime import datetime


def current_date():
    date = datetime.now().strftime("%Y/%m/%d
            %H:%M:%S")
    return date


# calling the function
print(current_date())
```

The *return* statement

- **The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.**

```
def add(x, y):
    print(x + y)

add(1, 2)
```

The *return* statement

- Can we return multiple values?

```
def fo(x,y):
    return x+1, y+1


print(fo(1, 2))
```

Function Arguments

- **Required arguments**
- **Keyword arguments**
- **Default arguments**
- **Variable-length arguments**

Required arguments

- Required arguments are the arguments passed to a function in correct positional order.

```
def info(name, age):
    print(name,'is',age)


info('william', 18)
```

Keyword arguments

- When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.

```
def info(name, age):
    print(name,'is',age)


info(name='william',age=18)
info(age=18,name='william')
```

Default arguments

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

```
def info(name, age=17):
    print(name,'is',age)
info('william')
info(name='william')
```

Variable-length arguments

- Variable-length arguments are not named in the function definition.
- Syntax:

```
def functionname(arg1, *args, **kwargs):
    function_suite
    return [expression]
```

Example

```
def printParams(data, *args):
    print('data:', data)
    print('args:', args)


printParams(10,20,30,40)
```

- *args is tuple that holds all a collection of nonkeyword arguments.

Example

```
def printParams2(**kwargs):
    print('kwargs:', kwargs)


printParams2(x=1,y=2,z=3)
```

- **kwargs is dictionary that hold a collection of keyword arguments

Python
Classes/instances

江琳

Philosophy

- **Python has been an object-oriented language since it existed. Because of this, creating and using classes and objects are downright easy.**

Creating a class

```
class Employee:
    pass
```

- Class is keyword used to define a class

Creating instances

```
# creating instances
emp1 = Employee()
```

Creating a class

```
class Employee:

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

emp1 = employee('Nancy', 5000)
```

- Passing two arguments to the class when creating a instance

Create a function in class

- Let create a function in Employee class that is used to print the name and salary

```
class employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        employee.count += 1

    def display_employee(self):
        print('name: ', self.name)
        print('salary: ', self.salary)
emp1 = employee('Nancy', 5000)
emp1.display_employee()
```

Creating a class

```
class employee:
    count = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        employee.count += 1

    def display_count(self):
        print('Total employee: ', employee.count)

    def display_employee(self):
        print('name: ', self.name)
        print('salary: ', self.salary)
```

Creating instances

```
# creating instances
emp1 = employee('Nancy', 5000)
emp2 = employee('Jessica', 6000)


# accessing attributes
emp1.display_employee()
emp1.display_count()
emp2.display_employee()
emp2.display_count()
```

Creating a class

- **The variable *count* is a class variable whose value is shared among all instances for a class**
- **__*init*__*()* is constructor or initialization method that Python calls when you create a new instance.**
- **You declare other class methods like normal functions with the exception that the first argument to each method is *self*.**

Python Modules

江琳

Philosophy

- **A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use.**
- **Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.**

Creating a module

- Create a file name current.py

```
from datetime import datetime


def current_date():
    date = datetime.now().strftime("%Y/%m/%d
            %H:%M:%S")
    return date
```

The *import* statement

- A module can be *imported* by another program to make use of its functionality.
- Open the terminal:
  - cd the directory where you store current.py

```
$python
>>> import current
>>> current.current.date()
```

The *from … import* statement

- Python's *from* statement lets you import specific attributes from a module into the current namespace.

```
$python
>>> from current import current_date
>>> current.date()
```

The *dir()* function

- **The dir() built-in function returns a sorted list of strings containing the names defined by a module.**

```
$python
>>> import current
>>> dir(current)
```

Build-in modules

- **A module can be *imported* by another program to make use of its functionality. This is how we can use the Python standard library as well.**
- **Now, we will see how to use the standard library modules.**

Build-in modules

```
from datetime import datetime
print(datetime.now())


import math
print(math.sin(5))


import uuid
print(uuid.uuid1())
```

Python file I/O

江琳

File I/O

- Python provides basic functions and methods necessary to manipulate files by default. You can do most of the file manipulation using a file object.

The *open* Function

- **Before you can read or write a file, you have to open it using Python‘s built-in *open()* function. This function creates a file object.**

```
file = open(file_name, [access_mode] ,[buffering])
```

The *open* Function

- ***file_name*: The file_name argument is a string value that contains the name of the file that you want to access.**
- ***access_mode*: The access_mode determines the mode in which the file has to be opened, i.e., read, write, append.**

The *open* Function

Modes	Description
r	Opens a file for reading only.
r+	Opens a file for both reading and writing.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	Opens a file for appending.

The *open* Function

- ***buffering*: If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default.**

The *close()* Method

- **The close() method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done.**

The *write()* Method

```
file = open('/Users/linjiang/Desktop/foo.txt', 'w')
file.write('Hello Python')
file.close()
```

The *read()* Method

```
file = open('/Users/linjiang/Desktop/foo.txt', 'r+')
print(file.read())
file.close()
```

The *readlines()* Method

- **Return a list.**

```
file = open('/Users/linjiang/Desktop/foo.txt', 'r+')
cxt = file.readlines()
for line in cxt:
    print(line)
file.close()
```

Questions?

Tutorial 2

Question 1

Write a program (using functions!) that asks the user for a long string containing multiple words. Print back to the user the same string, except with the words in backwards order. For example, say I type the string:

My name is Michele

Then I would see the string:

Michele is name My

shown back to me.

Question 2

Create a program that will play the “cows and bulls” game with the user. The game works like this:

Randomly generate a 4-digit number. Ask the user to guess a 4-digit number. For every digit that the user guessed correctly *in the correct place*, they have a “cow”. For every digit the user guessed correctly *in the wrong place* is a “bull.” Every time the user makes a guess, tell them how many “cows” and “bulls” they have. Once the user guesses the correct number, the game is over.

Question 2

Say the number generated by the computer is 1038. An example interaction could look like this:

```
Welcome to the Cows and Bulls Game!
Enter a number:
>>> 1234
2 cows, 0 bulls
>>> 1256
1 cow, 1 bull ...
```