Python Programming

Functions

Prof. Chang-Chieh Cheng
Information Technology Service Center
National Yang Ming Chiao Tung University

Functions

- Some pieces of code are useful and can be used again in the other places
- For example, computing the average of a list

```
L1 = [4, 5, 2, 1, 9]
avg1 = sum(L1) / len(L1)
print(avg1)
L2 = [0.4, 0.5, 0.2, 0.1, 0.9]
avg2 = sum(L2) / len(L2)
print(avg2)
```

- Do you need to write such reusable code again for using in next time?
  - No
  - Make it as a function!
  - Then call the function when you want to use it

Functions

- Syntax of a function definition

```
def function_name(parameter):
    function_code_block
```

def: abbreviation of **define**

- Output the result of a function
  - `return value`
  - For example,

```
def avg(L):                     # function definition
    return sum(L) / len(L)
                                # L is the parameter of avg

L1 = [4, 5, 2, 1, 9]
L2 = [0.4, 0.5, 0.2, 0.1, 0.9]
print(avg(L1))                  # function calling
print(avg(L2))                  # function calling

# L1 and L2 are arguments for function calling of avg
```

Functions

- Let's try it
  - Design a function named `median` that can find the median from a list.
  - So, the following program can be executed correctly.

```
L1 = [4, 5, 2, 1, 9]
L2 = [0.4, 0.5, 0.2, 0.1, 0.9]
print(median(L1))                         # 4
print(median(L2))                         # 0.4
```

Functions

- Function with multiple parameters
- An example, element-wise addition for two lists

```
def sumList(L1, L2):
  return [x + y for x, y in zip(L1, L2)]

L1 = [4, 5, 2, 1, 9]
L2 = [0.4, 0.5, 0.2, 0.1, 0.9]
L3 = sumList(L1, L2)
print(L3)                     # [4.4, 5.5, 2.2, 1.1, 9.9]
```

Functions

- Let's try it
  - append two parameters to `sumList`, `start` and `stop`, to indicate a data range of `L1` and `L2`.
  - Then, `sumList` returns a list that contains the result of element-wise addition of the specified range of `L1` and `L2`.
  - Try to let the following program can be executed correctly.

```
L1 = [4, 5, 2, 1, 9]
L2 = [0.4, 0.5, 0.2, 0.1, 0.9]
L3 = sumList(L1, L2, 0, 5)
print(L3)                      # [4.4, 5.5, 2.2, 1.1, 9.9]
L4 = sumList(L1, L2, 2, 4)
print(L4)                      # [2.2, 1.1]
L5 = sumList(L1, L2, 1, 4)
print(L5)                      # [5.5, 2.2, 1.1]
```

Function References

- A function can be refered by a variable
  - We can use a variable to store the memory address (link) of a function.
  - Aliasing a function name.

```
def f(x):
    return x * 10
def g(x):
    return x + 10

y = f(2)       # y stores the result of f(2)
print(y)
y = g(2)       # y stores the result of g(2)
print(y)

y = f          # y represents the function f
print(y)
z = y(3)       # calling f through y
print(z)
y = g          # y represents the function g
z = y(3)       # calling g through y
print(z)
```

Callback functions

- It is an application of function variable.
- Define a function to be an argument for another function.
- For example, define a special comparison rule for sorting

```
def f(s):
    return len(s)

Ls1 = ['cat', 'mouse', 'pig', 'dog', 'bird']
Ls2 = sorted(Ls1)
Ls3 = sorted(Ls1, key = f) # Passing f with key to sorted

print(Ls2)     # ['bird', 'cat', 'dog', 'mouse', 'pig']
print(Ls3)     # ['cat', 'pig', 'dog', 'bird', 'mouse']
```

Callback functions

- Let's try it
  - Modify the following code such that a list of numeric strings can be sorted by the numeric value of each string.
  - Try to let the following program can be executed correctly

```
def f(x):
    # ???

L1 = ['123', '000999', '54', '7.1', '   88']
L2 = sorted(L1, key =  f  )
print(L2)      # ['7.1', '54', '   88 ', '123', '000999']
```

Functions

- Default arguments

```
def addText(text1, text2, sep, end):
    return text1 + sep + text2 + end

s = addText('Hi', 'James', ' ', '.')
print(s)
s = addText('Hi', 'James')     # Error!
print(s)
```

```
def addText(text1, text2, sep = ' ', end = '.'):
    return text1 + sep + text2 + end

s = addText('Hi', 'James', '_', '?')
print(s)
s = addText('Hi', 'James')
print(s)
s = addText('Hi', 'James', '_')
print(s)
s = addText('Hi', 'James', , '?') # Error!
print(s)
```

Functions

- Keyword argument
  - Specify an argument by its parameter name.

```
def addText(text1, text2, sep = ' ', end = '.'):
    return text1 + sep + text2 + end

s = addText(text2 = 'James', end = '?', text1 = 'Hi!')
print(s)

s = addText(text2 = 'James', end = '?') # Error!
print(s)
```

Functions

- Let’s try it!
  - Design a function named `innerproduct`. It has four parameters that are
    - L1: list 1
    - L2: list2
    - start: the start index
    - stop: the stop index
  - Then, `innerproduct` can compute the inner product of two lists by the following equation:

$$L_1 \cdot L_2 = \sum_{i=start}^{stop-1} L_1[i] \times L_2[i]$$

```
La = [1, 2, 3, 4, 5]
Lb = [0.1, 0.2, 0.3, 0.4, 0.5]
x = innerproduct(La, Lb, 0, 5)
print(x)
x = innerproduct(La, Lb)
print(x)
x = innerproduct(La, Lb, stop = 3)
print(x)
```

Exercise 1

- Design a function, `leftpad(s, n, c)`
  - `s` and `c` are strings, `n` is a positive integer
- `leftpad` can padding a series of `c` to the left side of `s` such that the length of padded s is `n`.
- Try to let the following program can be executed correctly.

```
s = '1.234'
print(leftpad(s, 8, 'x'))      # xxx1.234
print(leftpad(s, 8))           # 0001.234
print(leftpad(s, 15, 'ABCD')) # CDABCDABCD1.234
print(leftpad(s, 0))           # 1.234
print(leftpad(n = 7, c = '@', s = s)) # @@1.234
```

- You can use range access in a string without any loop statement

```
s = 'ABCDEF'
print(s[1:3])   # BC
print(s[:3])    # ABC
print(s[2:])    # CDEF
```

Lambda functions

- A lambda function is a temporary function with a single expression
- Syntax:
  - **`lambda`** `parameter1, parameter2, …, parameterN:` **`expression`**
- We often design a lambda function to be a callback function.
- For example, define a special comparison rule for sorting

```
Ls1 = ['cat', 'mouse', 'pig', 'dog', 'bird']
Ls2 = sorted(Ls1)
Ls3 = sorted(Ls1, key = lambda x: len(x))
print(Ls2)     # ['bird', 'cat', 'dog', 'mouse', 'pig']
print(Ls3)     # ['cat', 'pig', 'dog', 'bird', 'mouse']
```

Lambda functions

- Let's try it
  - Modify the following code such that a list of numeric strings can be sorted by the numeric value of each string.
  - Try to let the following program can be executed correctly

```
L1 = ['123', '000999', '54', '7.1', '   88']
L2 = sorted(L1, key =  ???  )
print(L2)      # ['7.1', '54', '   88 ', '123', '000999']
```

Modules

- We can pack many function definitions into a .py file
- A module is a .py file containing Python definitions and statements
- For example, James.py contains four functions

```
def avg(L):
    # ...

def printList(L):
    # ...

def sumList(L1, L2, start = 0, stop = 0):
    # ...

def swap(a, b):
    # ...
```

Modules

- How to use a module?
- **import** module_name
- Usage
  - module_name.function

```
import James
L1 = [4, 5, 6, 7, 8]
L2 = [2, 3, 4 ,5 ,6]
print(James.avg(L1))
L3 = James.sumList(L1, L2)
L1, L2 = James.swap(L1, L2)
James.printList(L1)
```

Modules

- **import** `module_name` **as alias**

```
import James as J
L1 = [4, 5, 6, 7, 8]
L2 = [2, 3, 4 ,5 ,6]
print(J.avg(L1))
L3 = J.sumList(L1, L2)
L1, L2 = J.swap(L1, L2)
J.printList(L1)
```

Modules

- **from** module_name **import** item_name

```
from James import avg
L1 = [4, 5, 6, 7, 8]
L2 = [2, 3, 4 ,5 ,6]
print(avg(L1))                          # OK
L3 = sumList(L1, L2)                    # NameError
L1, L2 = James.swap(L1, L2)             # NameError
James.printList(L1)                     # NameError
```

Modules

- Let's try it
  - Design two functions to convert temperature between Fahrenheit and Celsius
    - `toC(F)`
      - Fahrenheit (ºF) to Celsius (ºC)
      - °C = (°F - 32) x 5/9
    - `toF(C)`
      - Celsius (ºF) to Fahrenheit(ºC)
      - °F = (°C × 9/5) + 32
  - Pack these functions into a module named `temperature`
  - Try to let the following program can be executed correctly

```
import temperature
print(temperature.toC(75.2))       # 24
print(temperature.toF(34.5))       # 94.1
```