Lecture 4 Function (2021.9.28 & 30)

§1 Function

1. Basic information

1° Some reusable code

2° take arguments as input, perform some computations, output some results

3° We call/invoke a function by using the function name, parenthesis and arguments

4° Types of functions

Built-in Functions:
- part of Python
- print(), int(), float(), etc.
- The names of built-in functions are usually considered as new reserved words

Self-Defined Functions:
- define by the programmers
- used by the programmers

2. Building a function

Using the def key word, followed by optional parameters in parenthesis

Program:

```
def Hello():
    print('Hello')
    print('Funny')

Hello()
print('Something in the middle.')
Hello()
```

indent the body of the function

Defines the function, but does NOT execute the body of the function

Output:

```
Hello
Funny
Something in the middle.
Hello
Funny
```

Program

```
x=5
print('Hello')

def print_lyrics():
    print('I am a lumberjack, and I am okay.')
    print('I sleep all night and I work all day.')

print('Yo')
x=x+2
print(x)
```

Output

```
Hello
Yo
7
```

Program

```
x=5
print('Hello')

def print_lyrics():
    print('I am a lumberjack, and I am okay.')
    print('I sleep all night and I work all day.')

print('Yo')
print_lyrics()
x=x+2
print(x)
```

Output

```
Hello
Yo
I am a lumberjack, and I am okay.
I sleep all night and I work all day.
7
```

3. Argument

1° An argument is a value we pass into the function as its input when we call the function

2° We use arguments so we can direct the function to do different kinds of work when we call it at different times.

3° We put the argument in the parenthesis after the name of the function.

```
print('I am the one')
```

argument

4. Parameters

1° A parameter is a variable

① used in the function definition

② is a 'handle' that allow the code in the function to access the arguments for a particular function invocation

- y = f(x) = x**2
  - x is a parameter
- f(3) = 3 ** 2 = 9
  - 3 is an argument

```
def greet(lang):
    if lang=='es':
        print('Hola')
    elif lang=='fr':
        print('Bonjour')
    else:
        print('Hello')

>>> greet('en')
Hello
>>> greet('es')
Hola
>>> greet('fr')
Bonjour
```

5. Return values

1° Often a function will return a value to be used as the value of the function call in the calling expression

2° Use the return keyword

Program

```
def greet():
    return 'Hello'

print(greet(),'Glenn')
print(greet(),'Sally')
```

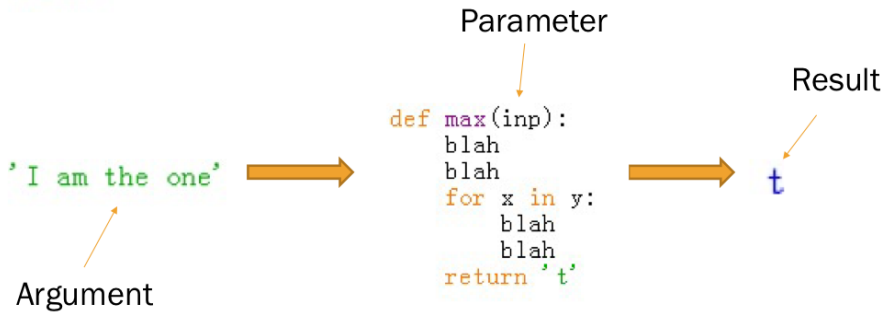
Output

```
Hello Glenn
Hello Sally
```

- A fruitful function is one that produces a result (or return value)
- The return statement ends the function execution and 'sends back' the result of the function

```
def greet(lang):
    if lang=='es':
        return 'Hola'
    elif lang=='fr':
        return 'Bonjour'
    else:
        return 'Hello'

>>> print(greet('en'),'Glenn')
Hello Glenn
>>> print(greet('es'),'Sally')
Hola Sally
>>> print(greet('fr'),'Michael')
Bonjour Michael
```

```
>>> big = max('I am the one')
>>> print(big)
t
```



6. Multiple parameters / arguments

1° We can define more than one parameter in a function definition.

2° We simply add more argument when we call the function

3° We match the number and order of arguments and parameters

```
def AddTwo(a, b):
    total = a+b
    return total

x=AddTwo(3, 5)
print(x)
```

7. Void functions

1° When a function doesn't return a value, it is called a "void" function.

2° It will return "None"

8. Scope of variables

The scope of a variable is the part of program where this variable can be accessed

1° Local variable: created inside a function and accessible

within the function.

2° Global variable: created outside a function and accessible to all functions.

```
globalVar = 1
def f1():
    localVar = 2
    print(globalVar)
    print(localVar)

f1()
print(globalVar)
print(localVar) # Out of scope, so this gives an error
```

3° global

Use keyword global to specify that a variable is a global variable

```
x = 1
def increase():
    global x
    x =  x + 1
    print(x) # Displays 2

increase()
print(x) # Displays 2
```

9. Default argument

1° Python allows you to define functions with default argument values

2° The default argument values will be passed to the function, when it is invoked without arguments

```
def printArea(width = 1, height = 2):
    area = width * height
    print("width:", width, "\theight:", height, "\tarea:", area)

printArea() # Default arguments width = 1 and height = 2
printArea(4, 2.5) # Positional arguments width = 4 and height = 2.5
printArea(height = 5, width = 3) # Keyword arguments width
printArea(width = 1.2) # Default height = 2
printArea(height = 6.2) # Default width = 1
```

10. Return Multiple values

1° Python allows a function to return multiple values

2° Pass the returned values in a simultaneous assignment.

```
def sort(number1, number2):
    if number1 < number2:
        return number1, number2
    else:
        return number2, number1

n1, n2 = sort(3, 2)
print("n1 is", n1)
print("n2 is", n2)
```

§2 Selected Build-in Functions

1. String type

1° A string is a sequence of characters

2° A string literal uses quotes ' ' or " ".

3° For strings, + means "concatenate"

4° When a string contains number, it is still a string.

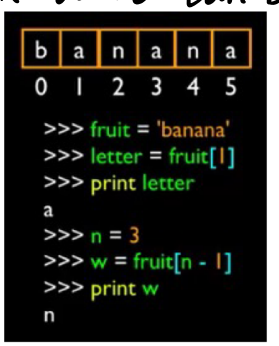
5° We can convert numbers in a string into a number using int( ) or float( )

2. Looking inside strings

1° We can get any character in a string using an index specified in square brackets

2° The index value must be an integer which starts from zero

3° The index value can be an expression



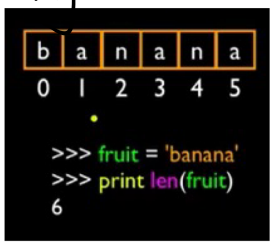
3. Max( ) and min( ) function

```
>>> big = max('Hello world')
>>> print(big)
w
```

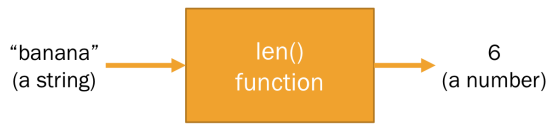
```
>>> small = min('Hello world')
>>> print(small)
```

4. len( )

There is a built-in function len( ) which gives us the length of a string




```
>>> fruit = 'apple'
>>> length = len(fruit)
>>> print(length)
5
```



5. Looping through strings

1° Using a for statement, we can easily loop through each character in a string

2° String is similar to a list in Python.

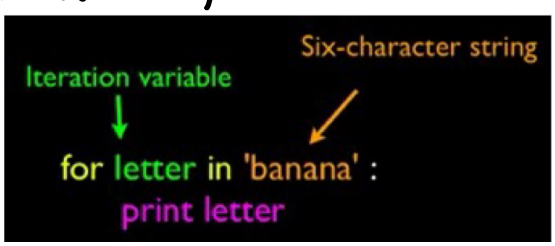
```
fruit = 'I am the one, Morpheus'

n = 0
for i in fruit:
    print(n, i)
    n=n+1

print('finished')
```

3° Loop and counting

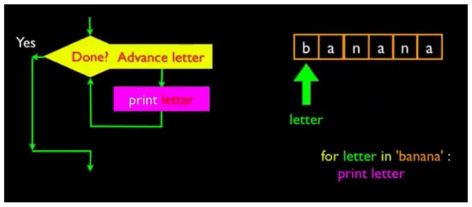
```
word = 'banana'
count = 0
for letter in word:
    if letter=='a':
        count = count+1
print("The number of 'a' we have seen is:", count)
```

6. Look deeper into in

1° The iteration variable "iterates" through the sequence (ordered set)



The iteration variable loops through the string, and the body of the loop is executed once for each character in that string



2° Using "in" in conditional statement

① The in keyword can also be used to check whether one string is in another string.

② The in expression is a logical expression and returns True or False

③ It can be used in if or while statement

```
>>> fruit = 'banana'
>>> 'n' in fruit
True
>>> 'm' in fruit
False
>>> 'nan' in fruit
True
>>> if 'a' in fruit:
        print('Got cha!')

Got cha!
```

3° Slicing strings

If we leave off the first or second number of the slice, it is assumed to be the beginning or end of the string respectively.


```
>>> s='Monty Python'
>>> print(s[:6])
Monty
>>> print(s[3:])
ty Python
>>> print(s[:])
Monty Python
```

7. File processing

1° Opening files

① This is done with the open( ) function

② open( ) returns a "file handle", a variable used to perform operations on files.

2° Using open( )

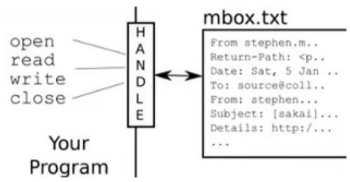
① handle = open(filename, mode)

② Returns a handle used to manipulate the file.

③ Filename is a string

④ Mode is optional, use "r" if we want to read the file, and "w" if we want to write to the file.

```
>>> fhand = open('c:\Python35\myhost.txt','r')
>>> print(fhand)
<_io.TextIOWrapper name='c:\\Python35\\myhost.txt' mode='r' encoding='cp936'>
```



3° When files are missing

```
>>> fhand = open('notExisting.txt','r')
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    fhand = open('notExisting.txt','r')
FileNotFoundError: [Errno 2] No such file or directory: 'notExisting.txt'
```

4° The newline character

① We use a new character to indicate when a line ends called "newline"

② We represent it as '\n' in strings

③ New line is still one character, not two

```
>>> stuff = 'Hello\nWorld'
>>> stuff
'Hello\nWorld'
>>> print(stuff)
Hello
World
>>> stuff = 'X\nY'
>>> print(stuff)
X
Y
>>> len(stuff)
3
```

5° File handle as a sequence

① A file handle open for reading can be treated as a sequence of strings where each line in the file is a string in sequence

② We can use the for statement to loop through a sequence

```
fhand = open('myhost.txt','r')

for line in fhand:
    print(line)

fhand.close()
```

Object Oriented Syntax

6° Reading the whole file

We can read the whole file into a single string

```
fhand = open('myhost.txt','r')
allText = fhand.read()
print('The length of the file:',len(allText))
print('The first 20 characters of the file:', allText[:20])
```

7° Searching through a file

We can put an if statement in the for loop to print the lines which satisfy certain conditions

```
fhand = open('myhost.txt','r')

for line in fhand:
    if line.startswith('#')==True:
        print(line)

print('finished.')
fhand.close()
```

Object Oriented Syntax

8° Writing to a file

To write a file, use the open() function with 'w' argument

Use the write() method to write to the file.

```
fhand = open('test.txt','w')
fhand.write('The first line\n')
fhand.write('The second line\n')
fhand.write('The third line\n')
fhand.close()
```

Object Oriented Syntax