Python Programming

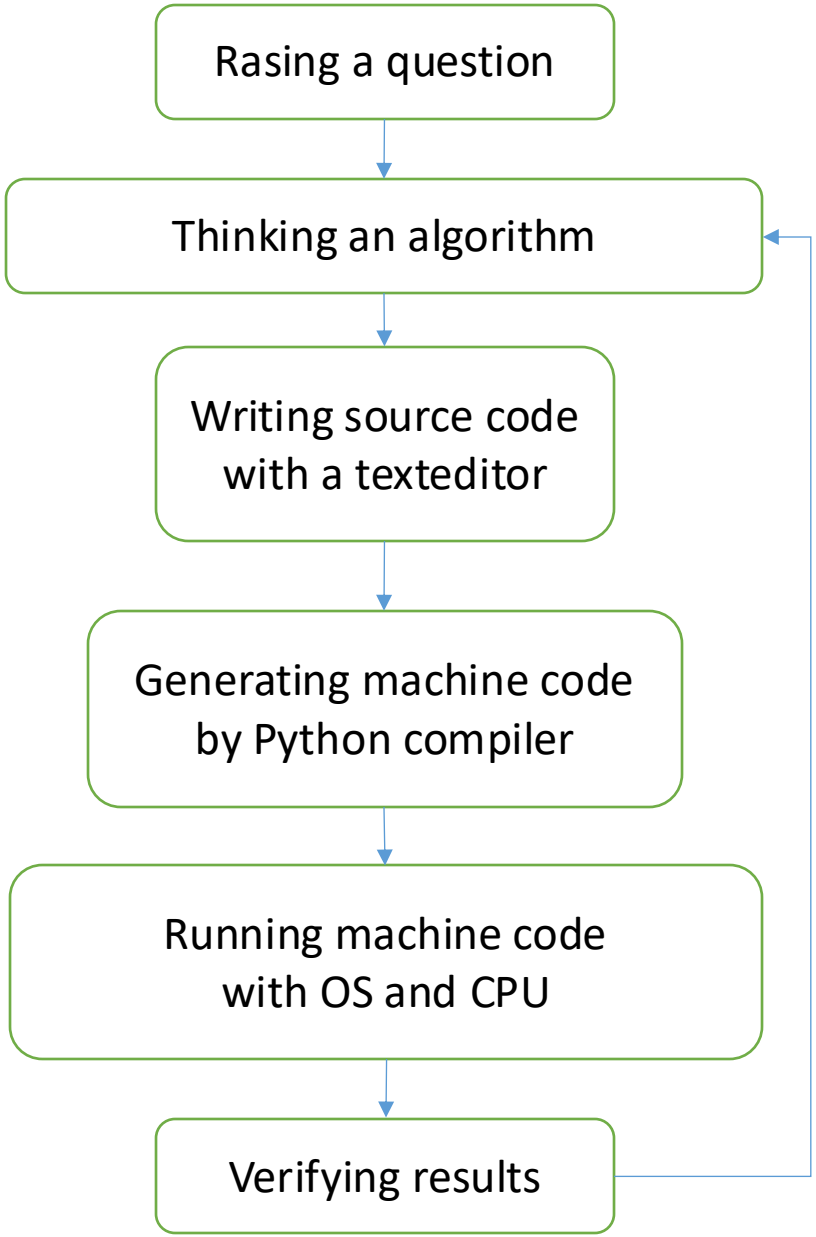
Getting Started

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

What is Programming?

- What is the main function of your computer?
- How to make your computer to do something for you?

Programming Flow



Simple Arithmetic

- What are the results of the following program?

```
print(1 + 2 + 3)
print(1 + 2 * 3 / 4)
```

Variables and Objects

- A variable or object can store a value for arithmetic.
  - A variable or object can provide a value in an arithmetical expression.
  - The value of a variable or object can be change.
- Run the following code:

```
x = 10
y = 2
print(x + y)
z = x / y
print(z)
x = x * z
y = y - z
print(x + y)
```

Variables and Objects

- Assignment operator "="
  - `a = b` means copying the value of `b` to `a`.
  - `a = b + c` means copying the value of `(b + c)` to `a`.
  - LHS, left-hand side
    - Creating variables
  - RHS, right-hand side
    - An expression can calculate values
  - Rule:
    1. RHS calculates **`n`** values
    2. LHS create **`n`** variables
    3. Assign the **`n`** values to the **`n`** variables, respectively.

Basic data type

- Data type
  - A data type indicates a group of data values with the same characteri
- Integers
  - Numbers without a fractional part
  - For example, 123, 0, and -51
  - Format symbol: %d
- Floats
  - Numbers that can have a decimal point, allowing them to represent fractions and more precise values than integers
  - For example, 3.14 and -2.5.
  - Format symbol: %f
- Strings
  - Sequences of characters used to represent text.
  - For example, "hello" and "12345"
  - Format symbol: %s

Strings

- Character
  - A unit of a text.
  - A letter, a numerical digit, or a symbol.
- String representation
  - Single quotes
    - `'ABC'`
    - `'123456890'`
  - Double quotes
    - `"ABC"`
    - `"1234567890"`
  - No different between single quotes and double quotes

Print Multiple Objects

- **`print`** is a function, which can display a message or data of a variable on screen.
  - **argument**: an input value of a function call.
  - Any two arguments are delimited by a comma.

```
t = 'Hello'
x = 5
y = 7
z = x / y
print(t, x, y, z)
print('%s'%t)
print('%d'%x)
print('%f'%z)
print('%s %d/%d=%.5f'%(t, x, y, z) )
```

  - **print** can output the values of arguments in the order from left to right.
  - Any two output results are delimited by a space

Separator Setting

- Comma separating

```
x = 10
y = 2
z = x / y
print(x, y, z, sep = ",")
```

- Text separating

```
x = 10
y = 2
z = x / y
print(x, y, z, sep = "@@@")
```

Strings

- Let's try it
  - Run the following program and see what results will be output.
  - Can you explain the reason of each output?

```
x = "XYZ"
y = 'ABC'
print(x, y)

x = "123"
y = '456'
z = x + y
print(z)

x = 123
y = 456
z = x + y
print(z)
```

```
x = "123"
y = 456
z = x + y
print(z)
```

Special Character

- Single quote

```
x = "\'"
print(x)
```

- Double quote

```
x = '\"'
print(x)
```

- tab

```
x = 'ABC\tXYZ'
print(x)
```

- newline

```
x = 'ABC\nXYZ'
print(x)
```

Separator and Terminal

- Tab separating

```
x = 10
y = 2
z = x / y
print(x, y, z, sep = "\t")
```

- Newline separating

```
x = 10
y = 2
z = x / y
print(x, y, z, sep = "\n")
```

Separator and Terminal

- Put a terminal text for each print.

```
x = 10
y = 2
z = x // y
print(x, end = " // ")
print(y, end = " = ")
print(z)
# 10 // 2 = 5
```

Separator and Terminal

- Let's try it:
  - Using `sep` and `end` to modify the following code:

```
x = 10
y = 2
z = x / y
w = x * y
print(x, y, z, w)
```

  - such that the result will be
    - `x■>>■y■>>■z■>>■w■[OK]`
    - where ■ is a white space.

Data Input

- **`input(prompt_string)`**
  - Read a string from standard input.
  - You can type data in IPython console window.
  - The trailing newline is stripped. (not including the newline character).

```
x = input("Input the first string: ")
print(x)
y = input("Input the second string: ")
print(x, y)
```

Data Conversion

- **`int(object)`**
  - Convert an object to an integer.

```
x = int(input("Input the first number: "))
y = int(input("Input the second number: "))
z = x / y
print(x, "/", y, "=", z)
```

  - You only can input a number without decimal; otherwise, you will get an error message:
    - **invalid literal for `int()`**

Data Conversion

- **`float(object)`**
  - Convert an object to a floating number (a real number).

```
x = float(input("Input the first number: "))
y = float(input("Input the second number: "))
z = x / y
print(x, "/", y, "=", z)
```

  - Therefore, you can input a number with decimal.

Comment in Python

- Comment
  - An explanation or annotation in the source code.
  - All comments will be ignored by Python interpreter.
- Single line comment `#`

```
# test
print(1 + 2 + 3)       # the result is 6
print(1 + 2 * 3 / 4)   # 2.5
```

- Multiple-line comment `"""` … `"""`

```
"""
This is my first Python program.
I love Python
very much!
"""
print(1 + 2 + 3)       # the result is 6
print(1 + 2 * 3 / 4)   # 2.5
```

Expressions

- An expression is a combination of one or more constants, variables, functions, and operators to produce values.

```
x = 1 - 2 * 3
y = abs(x) * 10
a, b = min(x, y), max(x, y)
print(a, b)
```

Operators

lowest precedence

highest precedence

Operator	Description
:=	Assignment expression
lambda	Lambda expression
if – else	Conditional expression
or	Boolean OR
and	Boolean AND
not x	Boolean NOT
in, not in, is, is not, <, <=, >, >=, !=, ==	Comparisons, including membership tests and identity tests
|	Bitwise OR
^	Bitwise XOR
&	Bitwise AND
<<, >>	Shifts
+, -	Addition and subtraction
*, @, /, //, %	Multiplication, matrix multiplication (numpy), division, floor division, remainder 5
+x, -x, ~x	Positive, negative, bitwise NOT
**	Exponentiation 6
await x	Await expression
x[index], x[index:index], x(arguments...), x.attribute	Subscription, slicing, call, attribute reference
(expressions...), [expressions...], {key: value...}, {expressions...}	Binding or parenthesized expression, list display, dictionary display, set display

Arithmetic Operators

- `+` addition `x + y`
- `-` subtraction `x - y`
- `*` Multiplication `x * y`
- `/` Division `x / y`
- `%` modulus
- `**` exponent
- `//` Floor division (integer division)

```
x = 11
y = 7
z = x % y
print(z)        # 4
z = y ** 2
print(z)        # 49
z = 2 ** 0.5
print(z)        # 1.4142135623730951
z = x / y
print(z)        # 1.5714285714285714
z = x // y
print(z)        # 1
```

Arithmetic Operators

- `//` Floor division
  - dividing and rounding down to the nearest integer.
  - `z = x // y`
    - `z` will be the nearest integer of `x / y` and smaller than `x / y`

```
x = 5
y = 2
z = x // y
print(z)        # 2

x = -5
y = 2
z = x // y
print(z)        # -3
```

```
x = 5
y = 2

a = x // y
b = -x // y
print(a, b)     # 2, -3

a = x // y
b = -a
print(a, b)     # ?, ?
```

Arithmetic Operators

- % Modulus
  - x % y
    - x - x // y * y

```
x = 11
y = 3
z = x % y       # 11 - (3 * 3)
print(z)        # 2

x = -11
y = 3
z = x % y       # -11 - (-4 * 3)
print(z)        # 1

x = 11
y = -3
z = x % y       # 11 - (-4 * -3)
print(z)        # -1
```

```
x = 11.4
y = 3
z = x // y      # 11.4 - (3 * 3)
print(z)        # 2.4

x = -11.4
y = 3
z = x // y      # -11.4 - (-4 * 3)
print(z)        # 0.6

x = 11.4
y = -3
z = x % y       # 11.4 - (-4 * -3)
print(z)        # -0.6
```

Arithmetic Assignment Operators

- `+=` `x += y` ➔ `x = (x + y)`
- `-=` `x -= y` ➔ `x = (x - y)`
- `*=` `x *= y` ➔ `x = (x * y)`
- `/=` `x /= y` ➔ `x = (x / y)`
- `%=` `x %= y` ➔ `x = (x % y)`
- `**=` `x **= y` ➔ `x = (x ** y)`
- `//=` `x //= y` ➔ `x = (x // y)`

```
x = 1
x += 1
print(x)        # 2
x *= x
print(x)        # 4
x %= 5
print(x)        # 4
x //= x - 1
print(x)        # 1
```

```
x = 1
x += x += 1     # Invalid syntax
x *= (x /= 1)   # Invalid syntax
```

Augmented Assignments

- **Augmented assignment operators**
  - `+=, -=, *=, ...`
  - `a += b` means copying the value of `b` to `a`.
  - `a = b + c` means copying the value of `(b + c)` to `a`.
  - LHS, left-hand side
    - An expression can create variables
    - LHS contains an undefined name will cause a `NameError`
  - RHS, right-hand side
    - An expression can calculate values
  - Rule:
    1. **Running LHS**
       Creating variables if LHS is an expression for variable creating.
    2. RHS calculates **`n`** values
    3. **Calculating the values of LHS and RHS with the augmented operator.**
    4. Assign the **`n`** values to the **`n`** variables, respectively.

Exercise 1

- Design a program for the simple coin change problem.
- Using `input()` to get two non-negative integers:
  - `price`
  - `payment`
    - `payment >= price`
- Finding the change to your customer.
  - Four types of coins in Taiwan:
    - 50 NTD, 10 NTD, 5 NTD, and 1 NTD.
  - Finding the change with the minimum number of coins.
    - For example:
      - `price = 17`
      - `payment = 500`
      - Then the change will be 483 and can be combined by
        - `50 * 9`
        - `10 * 3`
        - `5 * 0`
        - `1 * 3`

String Operators

- + String concatenation
- += String appending

```
x = 'james' + 'cheng' + 'cs'  # Concatenate three strings
print(x)                      # jameschengcs

y = x + '@''nctu.edu.tw'
        # + can be omiited for concatenating literal strings
print(y)                # jameschengcs@nctu.edu.tw

z = "email: "
z += y                  # Appending y to z
print(z)                # email: jameschengcs@nctu.edu.tw
```

Number to String

- `str(number)`

```
x = 123
y = 456
z = x + y
print(z)                    # 579
z = str(x) + str(y)
print(z)                    # 123456
```

- Let's try it:
  - Modify the fifth line, **`z = str(x) + str(y)`**, such that the result of the 6th line is

`123 + 456 = 579`

Lists

- Creating a list which can contain many objects
  - `listname = [object1, object2, …, objectN]`
- Accessing an item of a list
  - `listname[index]`
    - `index` is an integer.
  - The index of the first object in the list is **zero.**
    - zero-based indexing

```
L = [10, 20, 30, 4, 5, 6]
print(L[0])              # 10
print(L[3])              # 4
L[2] += L[4] + L[5]
print(L[2])              # 41
print(L)                 #[10, 20, 41, 4, 5, 6]
```

Lists

- An index can be negative.

```
L = [10, 20, 30, 4, 5, 6]      # N = 6
print(L[-1])    # ➔ L[N - 1] ➔ L[5] ➔ 6
print(L[-2])    # ➔ L[N - 2] ➔ L[4] ➔ 5
print(L[-6])    # ➔ L[N - 6] ➔ L[0]
```

- $-N \leq index < N$

```
L = [10, 20, 30, 4, 5, 6]      # N = 6
print(L[6])     # Out of range!
print(L[-7])    # ➔ L[N - 7] ➔ Out of range!
```

Lists

- The types of objects in a list can be different.

```
L = [10, 20, 30, 'ABC', '123', '456']
print(L[0])             # 10
print(L[3])             # ABC

L[0] += L[1] + L[2]
print(L[0])             # 60

L[3] += L[4] + L[5]
print(L[3])             # ABC123456

L[1] = L[4] + L[5]
print(L[1])             # 123456
                        # L[1] is changed to a string
```

Lists

- Be careful with the type error.

```
L = [10, 20, 30, 'ABC', '123', '456']

L[2] += L[4] + L[5]    # Type error!
                       # L[2] is an integer
                       # but L[4] + L[5] is a string
```

  - We will learn how to check the type of an object later.

- Let's try it:
  - `L = [10, 20, 30, 'ABC', '123', '456']`
  - Design a program to swap the first and last objects of `L`, such that the result of `print(L)` is

    `['456', 20, 30, 'ABC', '123', 10]`

Lists

- The length of a list
  - The number of items in a list
  - `len(list_object)`

```
L = [10, 20, 30, 'ABC', '123', '456']

print(len(L))  # 6
```

Lists

- Range accessing
  - `list[ S:T:D ]`
    - From `S` to `T`, `T` is not included, with an interval `D`.
    - The defaults values of `S`, `T`, and `D` are `0`, `N`, and `1` respectively.
    - `S < T` and the `S` and `T` must have the same sign; otherwise, the result is an empty list.

```
L = [10, 20, 30, 'ABC', '123', '456']
print( L[1:5:1] )       # [20, 30, 'ABC', '123']
print( L[1:5:2] )       # [20, 'ABC']
print( L[2:4] )         # Item 2 ~ Item 3
print( L[:3] )          # Item 0 ~ Item 2
print( L[3:] )          # Item 3 ~ Item N - 1
print( L[0:len(L)] )
print( L[-6:-1] )
print( L[:] )
print( L[::3])
```

Lists

- Range accessing

```
L = [10, 20, 30, 'ABC', '123', '456']
print(L[1:1])          # []
print(L[2:1])          # []
print(L[-1:-2])        # []
print(L[-2:3]) # []
```

Lists

- List operators
  - + list concatenation
  - += list appending

```
L1 = [10, 20, 30]
L2 = [40, 50, 60]
L3 = L1 + L2
print(L3)              # [10, 20, 30, 40, 50, 60]
L1 += L1
print(L1)              # [10, 20, 30, 10, 20 ,30]
```

Lists

- String can be regarded as a read-only list of characters.

```
s = 'ABCDEF'
print(s[0])     # A
print(s[3])     # D
```

- That means you **cannot** modify any character of a string.

```
s = 'ABCDEF'
s[2] = 'X'      # Error! each character is read-only!
```

- Range access in string:

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

Lists

- Converting a string to a character list.
  - `list(string_object)`
- Converting a character list to a string.
  - `str().join(list_object)`
    or
    `''.join(list_object)`

```
s = 'ABCDEF'
L = list(s)
print(L[0])     # A
print(L[3])     # D
L[2] = 'X'
print(L)        # ['A', 'B', 'X', 'D', 'E', 'F']
print(s)        # ABCDEF
s = ''.join(L)
print(s)        # ABXDEF
```

Assignment and List

- For integers and floats, the assignment is similar to data replication.

```
x = 1
y = x
y += 1
print(x)        # 1
print(y)        # 2

x = 0.5
y = x
y += 1
print(x)        # 0.5
print(y)        # 1.5
```

Assignment and List

- For other object, the assignment is similar to reference change (change the linking)

```
L1 = [1, 2, 3]
L2 = L1
L2[0] += 10
print(L1)       # [11, 2, 3]
print(L2)       # [11, 2, 3]
```

Assignment and List

- For string, the assignment is similar to reference change (change the linking).
  - However, string data is read-only, which means you cannot modify every character of a string

```
s1 = "hello"
s2 = s1
s2 = "abc"
print(s1)       # hello
print(s2)       # abc
```

Data Replication

- If you want to copy data from an object, you should call its constructor.

```
x = 1
y = int(x)      # copy the value of x to y

a = 0.5
b = float(a)    # copy the value of a to b

s1 = "hello"
s2 = str(s1)    # copy the value of s1 to s2

L1 = [1, 2, 3]
L2 = list(L1)   # copy the value of L1 to L2
L2[0] += 10
print(L1)       # [1, 2, 3]
print(L2)       # [11, 2, 3]
```

Exercise 2

- `L = [10, 20, 30, 'ABC', '123', '456']`
  - `If n is even`
  - `1st half part[0: n // 2]`
  - `2nd half part[n // 2: n]`
- Using the range accessing to swap the first part and second part of `L`, such that the result of `print(L)` is

  `['ABC', '123', '456', 10, 20, 30]`
- `L = [10, 20, 30, 'ABC', '123', '456', 'XYZ'] # Odd length`
  - `If n is odd`
  - `1st half part[0: n // 2+1]`
  - `2nd half part[n // 2+1: n]`
- Using the range accessing to swap the first part and second part of `L`, such that the result of `print(L)` is

  `['123', '456', 'XYZ', 10, 20, 30, 'ABC']`
- According the above method, can you write a program with range accessing to swap the first part and second part of **any list**?
- DO NOT use any iterative statement.