Lecture 2 Python Basics (2021.9.14 & 16)

§1 Python

1. What is Python?

Python is an interpred (解释型的), high-level, general-purpose programming language.

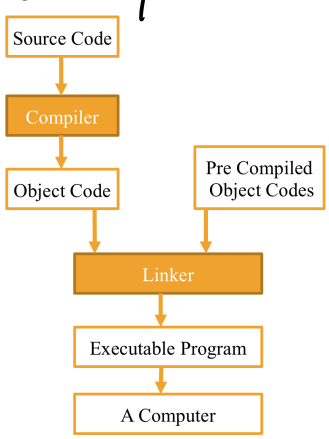
2. Compiler (编译器)

1° A computer can only understand low-level language.
High level languages can't be executed directly.

2° High level Languages must be translated into machine code to be executed.

3° Translated by a program called Compiler.

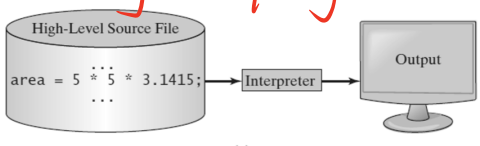
4° The output of the compiler is referred as Object Code



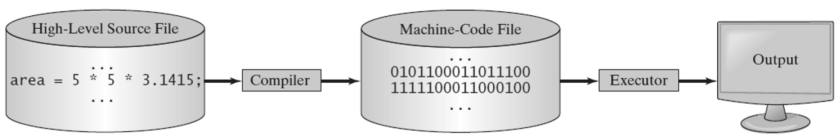
Compiling & Linking Programs
C++ Example

3. Interpreter (解释器,直译程序)

Interpreter is a computer program that directly executes source code without previously compiling into a machine code.



(a)



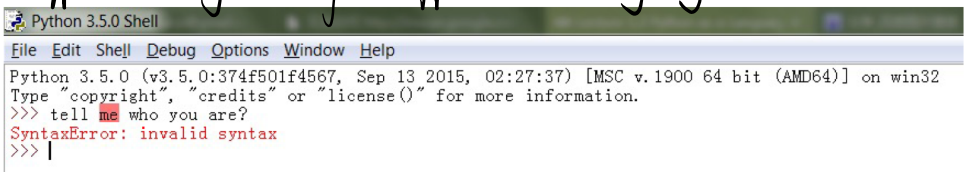
(b)

§2. What Should We Say to Python

1. Syntax

1° The rule of programming language

2° Different syntax for different languages.



2. Elements of Python Language

1° Vocabulary / Words

- Variables 变量
- Operators 运算符
- Reserved Words 保留字

2° Sentence Structure — valid syntax patterns

3° Story Structure — constructing a meaningful program for some purpose

3. Variables

1° Constants and variables

① Constants

Fixed value

- numbers
- letters
- strings
  - single-quotes (')
  - double-quotes (")

② Variables

(1) A variable is a named space in memory

(2) Referenced by the variable name determined by programmers.

(3) Operations on variable

- can store data
- can retrieve the data
- can change the data

2° Rules for defining Variables in Python

① Can only contain letters, numbers and underscore _

② Must start with a letter or underscore

③ Case sensitive

- Good: apple, car, myNumber123, _light
- Bad: 456aaa, #ab, var.12

* Personal Tips:

a. Use meaningful words as variable names

b. Start with a lower letter

c. Capitalize the first letter of each word

- Examples: myBankAccountID, numOfCards, salaryAtYear1995...
- Bad Examples:

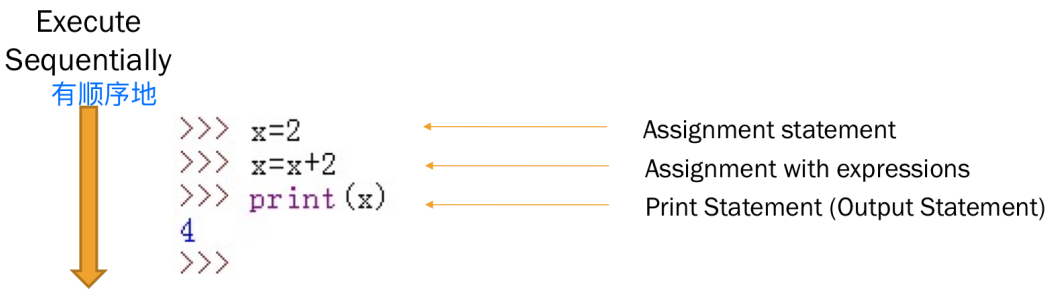
```
x1q3z9ocd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ocd * x1q3z9afd
print x1q3p9afd
```

4. Reserved Words

You cannot use the following words as variables.

```
False    None     True     and      as       assert   break
class    continue def      del      elif     else     except
finally  for      from     global   if       import   in
is       lambda   nonlocal not      or       pass     raise
return   try      while    with     yield
```

5. Sentences / Lines / Statements



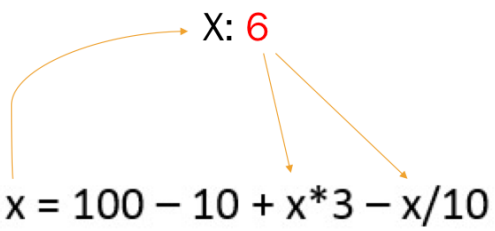
§3 Assignment

1. Assignment Statement

1° There is a location in the memory for x

2° Whenever the value of x is needed, it can be retrieved from the memory.

3° After the expression is evaluated, the result will be put back into x



2. Cascaded Assignment (级联赋值) (Not Recommended)

Set multiple variables into the same value using a single assignment statement

Example

```
>>> z = y = x = 2 + 7 + 2
>>> x, y, z
(11, 11, 11)
```

3. Simultaneous Assignment (同时赋值)

The values of two variables can be exchanged using simultaneous assignment.

Example

```
>>> c = "deepSecret"          # Set current password.
>>> o = "you'll never guess" # Set old password.
>>> c, o                      # See what passwords are.
('deepSecret', "you'll never guess")
>>> c, o = o, c               # Exchange the passwords.
```

```
>>> # A bad use of simultaneous assignment.
>>> x, y = (45 + 34) / (21 - 4), 56 * 57 * 58 * 59
>>> x, y
(4.647058823529412, 10923024)
>>> # A better way to set the values of x and y.
>>> x = (45 + 34) / (21 - 4)
>>> y = 56 * 57 * 58 * 59
>>> x, y
(4.647058823529412, 10923024)
```

4. Augmented Assignment

<lvalue> <op>= <expression>

Example

```
>>> x = 22        # Initialize x to 22.
>>> x += 7        # Equivalent to: x = x + 7
>>> x
29
>>> x -= 2 * 7    # Equivalent to: x = x - (2 * 7)
>>> x
15
```

§4. Evaluation Order of Expression

1. Operator Precedence

When we put operators together, Python needs to know which one to do first. This is called "operator precedence".

2. Numeric Expression and Operators

1° Asterisk (*) is the multiplication operators

2° Double asterisk (**) is used to denote Exponentiation (raise to a power)

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

3. Operator Precedence Rules

- Highest to Lowest Precedence Rule

1. Parenthesis 括号
2. Power
3. Multiplication, Division and Remainder
4. Addition and Subtraction
5. Left to Right

* Personal Tips

a. Use parenthesis

b. Keep mathematical expressions simple so that they are easy to understand.

c. Break up long series of math expressions to make them easy to understand

§5 Data Type

1. In Python, variables and constants have an associated "type"
   Python knows the difference between a number and a string

2. Type Matters

   1° Python knows what type everything is

   2° Some operations are prohibited on certain types

   3° You cannot "add 1" to a string

   4° We can check the type of something using function type( )

3. Type of Numbers

   1° Numbers in Python generally have two types:

   { Integers: 1, 2, 100, -20309 ...
   { Floating point numbers: 2.5, 3.7, 11.32309 ...

2° There are other number types, which are variations on the float and integer.

4. Floor Division (向下取整)

```
>>> time = 257           # Time in seconds.
>>> minutes = time // 60 # Number of complete minutes in time.
>>> print("There are", minutes, "complete minutes in", time, "seconds.")
There are 4 complete minutes in 257 seconds.
>>> 143 // 25
5
>>> 143.4 // 25
5.0
>>> 9 // 2.5
3.0
```

5. Divmond ( )

```
>>> time = 257        # Initialize time.
>>> SEC_PER_MIN = 60 # Use a "named constant" for 60.
>>> divmod(time, SEC_PER_MIN)  # See what divmod() returns.
(4, 17)
>>> # Use simulataneous assignment to obtain minutes and seconds.
>>> minutes, seconds = divmod(time, SEC_PER_MIN)
>>> # Attempt to display the minutes and seconds in "standard" form.
>>> print(minutes, ":", seconds)
4 : 17
>>> # Successful attempt to display time "standard" form.
>>> print(minutes, ":", seconds, sep="")
4:17
>>> # Obtain number of quarters and leftover change in 143 pennies.
>>> quarters, cents = divmod(143, 25)
>>> quarters, cents
(5, 18)
```

6. Type Can Change

A variable's type is determined by the value that is last assigned to the variable

```
>>> x = 7 * 3 * 2
>>> y = "is the answer to the ultimate question of life"
>>> print(x, y)         # Check what x and y are.
42 is the answer to the ultimate question of life
>>> x, y               # Quicker way to check x and y.
(42, 'is the answer to the ultimate question of life')
>>> type(x), type(y)   # Check types of x and y.
(<class 'int'>, <class 'str'>)
>>> # Set x and y to new values.
>>> x = x + 3.14159
>>> y = 12321213213123123123 12 * 98734237892374382 97
>>> print(x, y)         # Check what x and y are.
45.14159 12165255965071649871208683630735493412664
>>> type(x), type(y)   # Check types of x and y.
(<class 'float'>, <class 'int'>)
```

7. Type Conversion

When an expression contains both integer and float, integers will be converted into float implicitly

You can control this using functions int( ) and float( )

```
>>> print(float(99)/100)

>>> i=42
>>> type(i)

>>> f=float(i)
>>> print(f)

>>> type(f)

>>> print(1+2*float(3)-5)
```

8. String Conversions

You can also use int( ) and float( ) to convert strings into numbers
You will get an error if the string contains characters other than numbers

9. Converting Numbers into String

We can convert numbers into string using function str( )

```
>>> str(5)                # Convert int to a string.
'5'
>>> str(1 + 10 + 100)     # Convert int expression to a string.
'111'
>>> str(-12.34)           # Convert float to a string.
'-12.34'
>>> str("Hello World!")   # str() accepts string arguments.
'Hello World!'
>>> str(divmod(14, 9))    # Convert tuple to a string.
'(1, 5)'
>>> x = 42
>>> str(x)                # Convert int variable to a string.
'42'
```

§6 User Input

1. Input( )

We can instruct Python to stop and take user inputs using function input( )

```
name = input("Please input your name: ");
```

The input( ) function returns a string

2. Converting User Input

If we want to read a number using input( ), we must then

convert the input into a number using int() or float()

3. Comments

1° Anything after a "#" is ignored by Python.

2° The Use of comment

① Describe what is going to happen is a sequence of code

② Document who wrote the code and other important information

③ Turn off a line of code — usually temporarily

4. String Operations

Some operators apply to strings

1° "+": concatenation

2° "*": multiple concatenation

Python knows whether it is dealing with a number or a string.

§7 A Powerful Function — eval()

1. The eval() function

1° takes a string argument

2° evaluates that string as a Python expression

3° return the result of that expression

2. Caution: Users could potentially cause problems with "inappropriate" input.

EXAMPLE

```
>>> string = "5 + 12" # Create a string.
>>> print(string)      # Print the string.
5 + 12
>>> eval(string)       # Evaluate the string.
17
>>> print(string, "=", eval(string))
5 + 12 = 17
>>> eval("print('Hello World!')") # Can call functions from eval().
Hello World!
>>> # Using eval() we can accept all kinds of input...
>>> age = eval(input("Enter your age: "))
Enter your age: 57.5
>>> age
57.5
>>> age = eval(input("Enter your age: "))
Enter your age: 57
>>> age
57
>>> age = eval(input("Enter your age: "))
Enter your age: 40 + 17 + 0.5
>>> age
57.5
```

§8 List

1. What is list

A collection allows us to put many values in a single "variable"

2. List Constants

1° List constants are surrounded by square brackets and the elements in the list are separated by commas.

2° A list element can be any Python object — even another list.

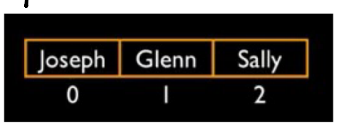
3° A list can be empty.

```
>>> print([1, 24, 76])
[1, 24, 76]
>>> print(['red','yellow','blue'])
['red', 'yellow', 'blue']
>>> print(['red',24, 98.6])
['red', 24, 98.6]
>>> print(1, [5, 6], 7)
1 [5, 6] 7
>>> print([])
[]
```

3. Looking inside lists

1° We can define list variables

2° We can access any single element in a list using an index specified in square bracket




```
>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> print(friends[1])
Glenn
```

4. Lists are mutable

We can change an element of a list using index operator

```
>>> lotto = [2, 14, 26, 41, 63]
>>> print(lotto)
[2, 14, 26, 41, 63]
>>> lotto[2]=28
>>> print(lotto)
[2, 14, 28, 41, 63]
```

§9 Tuple

1. Similar to List, but immutable
2. Created by a pair of parenthesis
3. Elements accessed by indices: variable_name [index]

Example:

test3.py

```
mylist = [1, 2, 3, 4, 'Oh']
mytuple = (1, 2, 'Hello', (4, 5))
print(mylist)
print(mytuple)
print(mylist[2])
print(mytuple[2])
```

Output:

```
[1, 2, 3, 4, 'Oh']
(1, 2, 'Hello', (4, 5))
3
Hello
```