Variable, expression, and statement

Table of Contents

1. First python program for BAIT, Arithmetic operators

(Chapter 1.4)

Python for Arithmetic

- Python provides operators, which are special symbols that represent computations like addition and multiplication.
- The operators +, -, * , and / perform addition, subtraction, and multiplication, as in the following examples:

```
>>> 10/2
5
>>> 10 + 2
12
>>> 10 * 10
100
```

- Exponentiation. To calculate 6^2+3

```
>>> 6**2 + 3
39

>>> 2 * 3 ** 2
18
```

- That operator % or modulo operator may not be very familiar to you. It does not return a percentage, but *remainder* after dividing one number from another.

```
>>> 10 % 2
0
```

```
>>> 10 % 4
2
```

2. Assignment statement and variables

(Chapter 2.1 and 2.2)

Assignment statements

- An assignment statement creates a new variable and gives it value. For example,

```
>>> n = 388
>>> n
388

>>> n - 3
385

>>> message = 'Hello World!'
>>> message
Hello World!
```

Variable names

- Variable names can be as long as you like. They can contain both letters and numbers.
- But they can't begin with a number.
- It is legal to use uppercase letters, but it is conventional to use only lowercase for variable names.
- The underscore character, _ , can appear in a name. It is often used in names with multiple words, such as *your_name* or *my_score*.
- If you give a variable an illegal name, you get a syntax error:

```
>>> 388course = 'Bus Program'
  File "<stdin>", line 1
    388course = 'Bus Program'
      ^
SyntaxError: invalid decimal literal
```

- `388course` is illegal because it begins with a number

```
>>> course@ = 388
  File "<stdin>", line 1
    course@ = 388
            ^
SyntaxError: invalid syntax
```

- `course@` is illegal because it contains an illegal character, @.

```
>>> class = 'bus Program'
  File "<stdin>", line 1
    class = 'bus Program'
          ^
SyntaxError: invalid syntax
```

- Variable name `class` is illegal because `class` is one of Python's keywords.
- The interpreter uses keywords to recognize the structure of the program, and they cannot be used as variable names.
- Python 3 has these keywords:

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

3. Interactive mode and script mode

(Chapter 2.4)

Interactive mode

- So far we have run Python in interactive mode, which means that you interact directly with the interpreter.
- In terminal, type `python` or `python3`, then `enter`. You will be present with `>>>` in the terminal window, where you can then run live, interactive code.

Script mode

- In VS code, we create a Python file with a filename extension `.py`
- For example, we create a file `calculater.py`. It contains the following code.

```
x = 10
y = 5

z = x * y
print(z)
```

- Then in terminal, we type `python3 calculator.py`

```
$ python3 calculator.py
50
```

- NOTE:
  - Before running `python3` command, you should make sure your current directory is where the file `calculator.py` is located. If not, use `cd` command to direct it.
  - In VS code, you can use go to `file->open folder` to open the folder that you would like to create the file or run the file. Then the terminal current directory will be exact the folder you would like to go to.

4. Values and types

(Chapter 1.5)

Integer or int

- Our first program

```
10/2
```

- The numbers here are **integer**. It is referred to as an `int`. We can use `type` function to see what type is the value or variable.

```
>>> type(10)
<class 'int'>
```

- For a variable,

```
>>> n = 388
>>> type(n)
<class 'int'>
```

Float

- Floating-point numbers, e.g., `10.0`, belong to `float`.

```
>>> type(10.0)
<class 'float'>>
```

- For a variable

```
>>> n = 3.14
>>> type(n)
<class 'float'>
```

String

- String value is the value with quotation marks.

```
>>> type('Hello')
<class 'str'>
```

- or

```
>>> message = 'Hello World!'
>>> type(message)
<class 'str'>
```

5. Strings

(Chapter 2.6 partly) **Values in Quotation marks**

- Actually, we can either use double quotation marks or single quotation marks.

```
>>> s = "hello"
>>> s
'hello'

>>> s = 'hello'
>>> s
'hello'
```

Small problem with quotation marks

- What if one of the character in the string is quotation mark?

```
>>> s = ""Hello!", he said."
  File "<stdin>", line 1
    s= ""Hello!", he said."
         ^
SyntaxError: invalid syntax
```

- There is an error, an invalid syntax. We cannot include *double quotation marks* in *double quotation marks*.
- Solutions

```
>>> s = "'Hello!', he said."
>>> s
"'Hello!', he said."

>>> s = '"Hello!", he said.'
>>> s
'"Hello!", he said.'

>>> s = "\"Hello!\", he said."
>>> s
'"Hello!", he said.'
```

String concatenation

- The + operator performs string concatenation, which means it joins the strings by linking them end-to-end. For example:

```
>>> s = 'Hello, '
>>> name = 'Python!'
>>> s + name
'Hello, Python!'
```

String Repetition

- The * operator also works on strings; it performs repetition. For example,

```
>>> s = 'Hello'
>>> s * 3
'HelloHelloHello'
```

String Built-in methods

- String Built-in methods: https://docs.python.org/3/library/stdtypes.html#string-methods

6. User input

- Please run the following codes in script mode.
- If we would like to say "hello" to someone, here is the code.

```
name = "Jin"
print("Hello,", name)
```

- It will be useful if we can allow users to input their own name, and make the program more interactive.
- `input` is a function that takes a prompt as an argument.

```
name = input("What's your name? ")
print("Hello,", name)
```

- Now let us do addition.

```
x = input("What's x? ")
y = input("What's y? ")

z = x + y

print(z)
```

Output

```
What's x? 1
What's y? 2
12
```

- It seems weird since the answer is expected to be 3. This is because our input from keyboard comes into Python as text. It is treated as type *string*.
- So we need to convert this input from string to an integer because we do addition. We should so as follows:

```
x = input("What's x? ")
y = input("What's y? ")

z = int(x) + int(y)

print(z)
```

- In the above program, the function `int()` helps convert string to integer.

7. String formatting

- If we run the following program.

```
name = 'Jin'
print('Hello, ' + name)
```

- In this program, we use + operator to concatenate two strings. The output is

```
Hello, Jin
```

- It is good. But if we would like to print a string together with an integer, the concatenation operation doesn't work. For example,

```
n = 35
print("The number of students is " + n)
```

- It generates the following error, which says that Python can only concatenate string (not "int") to string.

```
    print("The number of students is " + n)
          ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^~~
TypeError: can only concatenate str (not "int") to str
```

- We may convert int to string by using `str()` function first, then do concatenation, like this

```
n = 35
n_as_str = str(n)
print("The number of students is " + n_as_str)
```

- Seems good. However, this kind of conversion could be annoying. For example, if we would like to print a sentence with more number of int.

```
x = 3
y = 4

z = x + y

# convert x, y, z to str
x_as_str = str(x)
y_as_str = str(y)
z_as_str = str(z)

# print
print(x_as_str + " plus " + y_as_str + " equals " + z_as_str)
```

- The above program works, but it is really not elegant. First, it is boring to convert all the integers to strings. Second, there are so many + symbols that we cannot clearly see what we are printing.

- Fortunately, in Python 3.6 and above, it gives as the *f-strings*, which make the string formatting much easier. We can do it like this.

```
x = 3
y = 4

z = x + y

print(f"{x} plus {y} equals {z}.")
```

Summing up

- Arithmetic operators
- Variables
- Terminal
- `.py` file
- Types, int, float, and string
- String concatenation and repetition
- User input
- f-strings