4. Python Basic Syntaxes

4.1 Comments

A Python comment begins with a hash sign (#) and last till the end of the current line. Comments are ignored by the Python Interpreter, but they are critical in providing explanation and documentation for others (and yourself three days later) to read your program. Use comments liberally.
There is NO multi-line comment in Python?! (C/C++/Java supports multi-line comments via /* ... */.)

4.2 Statements

A Python statement is delimited by a newline. A statement cannot cross line boundaries, except:

1. An expression in parentheses (), square bracket [], and curly braces {} can span multiple lines.
2. A backslash (\) at the end of the line denotes continuation to the next line. This is an old rule and is NOT recommended as it is error-prone.

Unlike C/C++/C#/Java, you don't place a semicolon (;) at the end of a Python statement. But you can place multiple statements on a single line, separated by semicolon (;). For examples,

```
# One Python statement in one line, terminated by a newline.
# There is no semicolon at the end of a statement.
>>> x = 1     # Assign 1 to variable x
>>> print(x)  # Print the value of the variable x
1
>>> x + 1
2
>>> y = x / 2
>>> y
0.5

# You can place multiple statements in one line, separated by semicolon.
>>> print(x); print(x+1); print(x+2)  # No ending semicolon
1
2
3

# An expression in brackets [] can span multiple lines
>>> x = [1,
      22,
      333]  # Re-assign a list denoted as [v1, v2, ...] to variable x
>>> x
[1, 22, 333]

# An expression in braces {} can also span multiple lines
>>> x = {'name':'Peter',
      'gender':'male',
      'age':21
     }  # Re-assign a dictionary denoted as {k1:v1, k2:v2,...} to variable x
>>> x
{'name': 'Peter', 'gender': 'male', 'age': 21}

# An expression in parentheses () can also span multiple lines
# You can break a long expression into several lines by enclosing it with parentheses ()
>>> x =(1 +
     2
```

```
    + 3
    -
    4)
>>> x
2

# You can break a long string into several lines with parentheses () too
>>> s = ('testing '   # No commas
     'hello, '
      'world!')
>>> s
'testing hello, world!'
```

4.3 Block, Indentation and Compound Statements

A block is a group of statements executing as a unit. Unlike C/C++/C#/Java, which use braces {} to group statements in a body block, Python uses indentation for body block. In other words, indentation is syntactically significant in Python - the body block must be properly indented. This is a good syntax to force you to indent the blocks correctly for ease of understanding!!!

A compound statement, such as conditional (if-else), loop (while, for) and function definition (def), begins with a header line terminated with a colon (:); followed by the indented body block, as follows:

```
header_1:         # Headers are terminated by a colon
   statement_1_1  # Body blocks are indented (recommended to use 4 spaces)
   statement_1_2
   ......
header_2:
   statement_2_1
   statement_2_2
   ......

# You can place the body-block in the same line, separating the statement by semi-colon (;)
# This is NOT recommended.
header_1: statement_1_1
header_2: statement_2_1; statement_2_2; ......
```

For examples,

```
# if-else
x = 0
if x == 0:
   print('x is zero')
else:
   print('x is not zero')

# or, in the same line
if x == 0: print('x is zero')
else: print('x is not zero')

# while-loop sum from 1 to 100
sum = 0
number = 1
while number <= 100:
   sum += number
   number += 1
```

```
print(sum)

# or, in the same line
while number <= 100: sum += number; number += 1

# Define the function sum_1_to_n()
def sum_1_to_n(n):
   """Sum from 1 to the given n"""
   sum = 0;
   i = 0;
   while (i <= n):
      sum += i
      i += 1
   return sum

print(sum_1_to_n(100))  # Invoke function
```

Python does not specify how much indentation to use, but all statements of the SAME body block must start at the SAME distance from the right margin. You can use either space or tab for indentation but you cannot mix them in the SAME body block. It is recommended to use 4 spaces for each indentation level.

The trailing colon (:) and body indentation is probably the most strange feature in Python, if you come from C/C++/C#/Java. Python imposes strict indentation rules to force programmers to write readable codes!

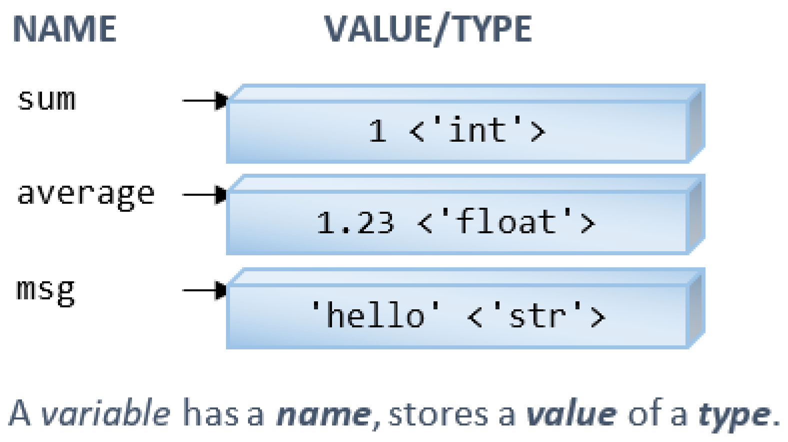
4.4 Variables, Identifiers and Constants

Like all programming languages, a variable is a *named* storage location. A variable has a name (or *identifier*) and holds a value.

Like most of the scripting interpreted languages (such as JavaScript/Perl), Python is dynamically typed. You do NOT need to declare a variable before using it. A variables is created via the initial assignment. (Unlike traditional general-purpose static typed languages like C/C++/Java/C#, where you need to declare the name and type of the variable before using the variable.)

For example,

```
>>> sum = 1         # Create a variable called sum by assigning an integer into it
>>> sum
1
>>> type(sum)       # Check the data type
<class 'int'>
>>> average = 1.23   # Create a variable called average by assigning a floating-point number into it
>>> average
1.23
>>> average = 4.5e-6  # Re-assign a floating-point value in scientific notation
>>> average
4.5e-06
>>> type(average)     # Check the data type
<class 'float'>
>>> average = 78      # Re-assign an integer value
>>> average
78
>>> type(average)     # Check the data type
<class 'int'>       # Change to 'int'
>>> msg = 'Hello'    # Create a variable called msg by assigning a string into it
>>> msg
'Hello'
>>> type(msg)        # Check the data type
<class 'str'>
```



As mentioned, Python is dynamic typed. Python associates types with the objects, not the variables, i.e., a variable can hold object of any types, as shown in the above examples.

Rules of Identifier (Names)

An identifier starts with a letter (A-Z, a-z) or an underscore (_), followed by zero or more letters, underscores and digits (0-9). Python does not allow special characters such as $ and @.

Keywords

Python 3 has 35 reserved words, or keywords, which cannot be used as identifiers.

- True, False, None (boolean and special literals)
- import, as, from
- if, elif, else, for, in, while, break, continue, pass, with (flow control)
- def, return, lambda, global, nonlocal (function)
- class
- and, or, not, is, del (operators)
- try, except, finally, raise, assert (error handling)
- await, async, yield

Variable Naming Convention

A variable name is a noun, or a noun phrase made up of several words. There are two convenctions:

1. In lowercase words and optionally joined with underscore if it improves readability, e.g., num_students, x_max, myvar, isvalid, etc.
2. In the so-called camel-case where the first word is in lowercase, and the remaining words are initial-capitalized, e.g., numStudents, xMax, yMin, xTopLeft, isValidInput, and thisIsAVeryLongVariableName. (This is the Java's naming convention.)

Recommendations

1. It is important to choose a name that is *self-descriptive* and closely reflects the meaning of the variable, e.g., numStudents, but not n or x, to store the number of students. It is alright to use abbreviations, e.g., idx for index.
2. Do not use *meaningless* names like a, b, c, i, j, k, n, i1, i2, i3, j99, exercise85 (what is the purpose of this exercise?), and example12 (What is this example about?).
3. Avoid *single-letter* names like i, j, k, a, b, c, which are easier to type but often meaningless. Exceptions are common names like x, y, z for coordinates, i for index. Long names are harder to type, but self-document your program. (I suggest you spend sometimes practicing your typing.)
4. Use *singular* and *plural* nouns prudently to differentiate between singular and plural variables. For example, you may use the variable row to refer to a single row number and the variable rows to refer to many rows (such as a list of rows - to be discussed later).

Constants

Python does not support constants, where its contents cannot be modified. (C supports constants via keyword const, Java via final.)
It is a convention to name a variable in uppercase (joined with underscore), e.g., MAX_ROWS, SCREEN_X_MAX, to indicate that it should not be modified in the program. Nevertheless, nothing prevents it from being modified.

4.5 Data Types: Number, String and List

Python supports various number type such as int (for integers such as 123, -456), float (for floating-point number such as 3.1416, 1.2e3, -4.5E-6), and bool(for boolean of either True and False).
Python supports text string (a sequence of characters). In Python, strings can be delimited with single-quotes or double-quotes, e.g., 'hello', "world", '' or ""(empty string).
Python supports a dynamic-array structure called list, denoted as *lst* = [*v1*, *v2*, ..., *vn*]. You can reference the i-th element as *lst*[*i*]. Python's list is similar to C/C++/Java's array, but it is NOT fixed size, and can be expanded dynamically during runtime.

I will describe these data types in details in the later section.

4.6 Console Input/Output: input() and print() Built-in Functions

You can use built-in function input() to read input from the console (as a string) and print() to print output to the console. For example,

```
>>> x = input('Enter a number: ')
Enter a number: 5
>>> x
'5'        # A quoted string
>>> type(x)  # Check data type
<class 'str'>
>>> print(x)
5

# Cast input from the 'str' input to 'int'
>>> x = int(input('Enter an integer: '))
Enter an integer: 5
>>> x
5          # int
>>> type(x)  # Check data type
<class 'int'>
>>> print(x)
5
```

print()

The built-in function print() has the following signature:

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
   # Print objects to the text stream file (default standard output sys.stdout),
   #   separated by sep (default space) and followed by end (default newline).
```

For examples,

```
>>> print('apple')  # Single item
apple
>>> print('apple', 'orange')  # More than one items separated by commas
apple orange
>>> print('apple', 'orange', 'banana')
apple orange banana
```

print()'s separator (sep) and ending (end)

You can use the optional *keyword-argument* sep='x' to set the separator string (default is space), and end='x' for ending string (default is newline). For examples,

```
# print() with default newline
>>> for item in [1, 2, 3, 4]:
       print(item)  # default is newline
1
2
3
4
# print() without newline
>>> for item in [1, 2, 3, 4]:
       print(item, end='')   # suppress end string
1234
# print() with some arbitrary ending string
>>> for item in [1, 2, 3, 4]:
      print(item, end='--')
1--2--3--4--
# Test separator between items
>>> print('apple', 'orange', 'banana')  # default is space
apple orange banana
>>> print('apple', 'orange', 'banana', sep=',')
apple,orange,banana
>>> print('apple', 'orange', 'banana', sep=':')
apple:orange:banana
>>> print('apple', 'orange', 'banana', sep='|')
apple|orange|banana
>>> print('apple', 'orange', 'banana', sep='\n')  # newline
apple
orange
banana
```

print in Python 2 vs Python 3

Recall that Python 2 and Python 3 are NOT compatible. In Python 2, you can use "print item", without the parentheses (because print is a keyword in Python 2). In Python 3, parentheses are required as print() is a function. For example,

```
# Python 3
>>> print('hello')
hello
>>> print 'hello'
  File "<stdin>", line 1
    print 'hello'
              ^
SyntaxError: Missing parentheses in call to 'print'
>>> print('aaa', 'bbb')
aaa bbb
   # Treated as multiple arguments, printed without parentheses

# Python 2
>>> print('Hello')
Hello
>>> print 'hello'
hello
>>> print('aaa', 'bbb')
('aaa', 'bbb')
   # Treated as a tuple (of items). Print the tuple with parentheses
>>> print 'aaa', 'bbb'
aaa bbb
   # Treated as multiple arguments
```

Important: Always use print() function with parentheses, for portability!