CSCI2040: Introduction to Python: Variables, Strings, and Numbers

Yim-Pan CHUI

Computer Science & Engineering Department

The Chinese University of Hong Kong

Variable

- Let us try the famous "hello world"

```python
print("Hello world")

```

```
Hello world
```

- Create and assign variable

```python
message = "Hello to the wonderful world of Python"
print(message)
```

```
Hello to the wonderful world of Python
```

Variable

- Let us try the famous "hello world"

```python
message2 = "and welcome to CSCI2040"
print(message)
print(message2)
```

```
Hello to the wonderful world of Python
and welcome to CSCI2040
```

```python
print(message, message2)  # try to print both messages in one line

```

```
Hello to the wonderful world of Python and welcome to CSCI2040
```

Comments

- # This line is a comment.
- Note that # can also be put after a computational statement

Good practices for comments

- It is short and to the point, but a complete thought.
- It explains your thinking, so that when you return to the code later you will understand
- how you were approaching the problem.
- It explains your thinking, so that others who work with your code will understand your overall approach.
- It explains particularly difficult sections of code in detail.

Naming rules

1. Variables can only contain letters, numbers, and underscores. Variable names can start with a letter or an underscore, but can not start with a number.
2. Spaces are not allowed in variable names, so we use underscores instead of spaces. For example, use student_name instead of "student name".
3. You cannot use Python keywords as variable names.

- Guidelines
- Variable names should be descriptive, without being too long. For example mc_wheels is better than just "wheels", and number_of_wheels_on_a_motorcycle.
- Be careful about using the lowercase letter l and the uppercase letter O in places where they could be confused with the numbers 1 and 0.

Reserved Words

- *Reserved words* or *keywords* are names that have special meaning in Python.

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	

strings

```
my_string1 = "a string with a double quote"
my_string2 = 'a string with a single quote'
print ("STR1=", my_string1)
print ('STR2=', my_string2) # let's see the output
```

```
STR1= a string with a double quote
STR2= a string with a single quote
```

- *what about a string which contains a quote?*

```
quote = "Martin Luther King Jr. said, 'Free at last, free at last.'"
print (quote)
```

```
Martin Luther King Jr. said, 'Free at last, free at last.'
```

Changing cases for the string

```python
first_name = 'go'
print(first_name) # just print the string
print(first_name.title()) # capitalize the first letter
print(first_name.upper()) # capitalize the whole string
```

```
go
Go
GO
```

- *Lower case as well*

```python
last_name = "Lamp"
print(last_name.lower())
```

```
lamp
```

Combine strings - Concatenation

```
print ("Our king's first name is =", first_name)
print ("Our king's last name is =", last_name)
full_name = first_name + last_name
print ("Our king's name is = ", full_name)
```

```
Our king's first name is = go
Our king's last name is = Lamp
Our king's name is =  goLamp
```

- *Lower case as well*

```
full_name1 = first_name.title() + " " + last_name
print ("Our king's full name is = ", full_name1)
```

```
Our king's full name is = Go Lamp
```

Adding control characters into strings

```
print(full_name1, "is a jerk") # automatically add one space
print(full_name1, "\tis a jerk") # adding a tab
print(full_name1, "\t\t\tis a jerk!!!!") # adding 3 tabs

```

```
Go Lamp is a jerk
Go Lamp 	is a jerk
Go Lamp 			is a jerk!!!!
```

- *New line*

```
print(full_name1, "\nis a jerk")
print("\n",full_name1, "\n\t\tis a jerk")
```

```
Go Lamp
is a jerk

 Go Lamp
		is a jerk
```

Stripping white spaces

- In user input, they may inadvertently add more "white spaces" than they intended to.

```
name = " CUHK " # a string with white spaces, before and after
print("stripping left of the string, name=" + " ***" + name.lstrip() + "***")
print("stripping right of the string, name=" + " ***" + name.rstrip() + "***")
print("stripping both sides of the string, name=" + " ***" + name.strip() + "***")
```

```
stripping left of the string, name= ***CUHK ***
stripping right of the string, name= *** CUHK***
stripping both sides of the string, name= ***CUHK***
```

Strings

- **type() and dir()**

```
stuff = 'hello world'
type(stuff)
dir(stuff) # show all built-in functions of tyis type
help(stuff.capitalize) # show the meaning of the capitalize function
```

```
<class 'str'>
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isdecimal',
'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper',
'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title',
'translate', 'upper', 'zfill']
Help on built-in function capitalize:

capitalize(...) method of builtins.str instance
    S.capitalize() -> str

    Return a capitalized version of S, i.e. make the first character
    have upper case and the rest lower case.
```

Numbers and numerics

- **Integers**

```
print (1+2.) # addition
print (3.-2) # subtraction
print (5*2.) # multiplication
print (8/4) # division
print (2.0**4) # exponentiation
print (2+3*4)
print ((2+3)*4)
```

```
3.0
1.0
10.0
2.0
16.0
14
20
```

Numbers and numerics

- **Floating points**

```
print (0.1+0.1)
print (0.1+0.2)
```

```
0.2
0.30000000000000004
```

Division in Python 2.7 and Python 3.X

- In Python 2.7; print 3/2 will give you 1
- In Python 3.X, print 3/2 will give you 1.5

- There are more differentiation between 3.X and 2.X for Python

operators: +, -, , *, /, //, %

```python
print(3+5) # adding two integers
print(3. + 5) # type conversion
print('Aaa' + 'Bbbbbb' +'CCC') # concatenation
```

```
8
8.0
AaaBbbbbbCCC
```

```
print(-5.2) # gives a negative number
print(5-3)
print(2*3)
print('Lu'*2) # generate character strings
print(3**4) # power
print (3.0**4.)
print(5/3)
print(5/3.)
print(4//3) # gives floor
print(5.0//3.)
print (17 % 3) # gives remainder
print (17. % 3)
```

```
-5.2
2
6
LuLu
81
81.0
1.6666666666666667
1.6666666666666667
1
1.0
2
2.0
```

Operator	Description	Examples
<	Less Than	5 < 3 gives 0 (i.e. False) and 3 < 5 gives 1 (i.e. True). Comparisons can be chained arbitrarily: 3 < 5 < 7 gives True.
>	Greater Than	5 > 3 returns True. If both operands are numbers, they are first converted to a common type. Otherwise, it always returns False.
<=	Less Than or Equal To	x = 3; y = 6; x <= y returns True.
>=	Greater Than or Equal To	x = 4; y = 3; x >= 3 returns True.
==	Equal To	x = 2; y = 2; x == y returns True. x = 'str'; y = 'stR'; x == y returns False. x = 'str'; y = 'str'; x == y returns True.
!=	Not Equal To	x = 2; y = 3; x != y returns True.
not	Boolean NOT	x = True; not y returns False.
and	Boolean AND	x = False; y = True; x and y returns False since x is False. In this case, Python will not evaluate y since it knows that the value of the expression will has to be false (since x is False). This is called short-circuit evaluation.
or	Boolean OR	x = True; y = False; x or y returns True. Short-circuit evaluation applies here as well.

If statement

- Selection statement for 1 or more choices

```
if choice == 1:
    print ('You choose Cola')
    out = 'coke'
elif choice == 2:
    print ('You choose Lemon tea')
    out = 'Lemon Tea'
elif choice == 3:
    print ('You choose Orange juice')
    out = 'Orange Juice'
else:
    print ('Invalid choice!')
    print ('choose again')
```

Only one outcome is selected

If statement

- Selection statement for 1 or more choices

```
if choice == 1:
    print ('You choose Cola')
    out = 'coke'
elif choice == 2:
    print ('You choose Lemon tea')
    out = 'Lemon Tea'
elif choice == 3:
    print ('You choose Orange juice')
    out = 'Orange Juice'
else:
    print ('Invalid choice!')
    print ('choose again')
```

Note the : after each Condition

Also statements indented are executed for each choice

If statement

- All alternatives i.e. elif & else are optional.

```
if score >= 50:
    print ('Passed')
```

- Or

```
if score >= 50:
    print ('Passed!')
else:
   print ('Failed!')
print ('Prepare for next test.')
```

Control Flow

```
a, b = 0, 1
while b < 10:
    print (b)
    a, b = b, a+b
print ('END')
```

Output

```
1
1
2
3
5
8
END
```

Control Flow

```
a, b = 0, 1
while b < 10:
    print (b)
    a, b = b, a+b
print ('END')
```

Multiple assignments
a = 0
b = 1

Assign at the same time

Control Flow

```
a, b = 0, 1
while b < 10:
    print (b)
    a, b = b, a+b
print ('END')
```

Keep doing statements in red box as long as condition is true

Note the ':' a must

Control Flow

```
a, b = 0, 1
while b < 10:
    print (b)
    a, b = b, a+b
print ('END')
```

Statements defined by indentation

Control Flow

```
a, b = 0, 1
while b < 10:
    print (b)
    a, b = b, a+b
print ('END')
```

First time : b = 1
Enter loop
print b

Output

```
1
```

Control Flow

```
a, b = 0, 1
while b < 10:
    print (b)
    a, b = b, a+b
print ('END')
```

Using old values of a & b

First time : b = 1

a update to 1
b update to 0+1

Note: the two assignment update at the same time

Control Flow

```
a, b = 0, 1
while b < 10:
    print (b)
    a, b = b, a+b
print ('END')
```

Program flow loop to while and check condition again

Control Flow

```
a, b = 0, 1
while b < 10:
    print (b)
    a, b = b, a+b
print ('END')
```

2nd time : b = 1

Enter loop

print b

Output

```
1
1
```

Control Flow

```
a, b = 0, 1
while b < 10:
    print (b)
    a, b = b, a+b
print ('END')
```

Using old values of a & b

2nd time : b = 1

a update to 1
b update to 1+1

Control Flow

```
a, b = 0, 1
while b < 10:
    print (b)
    a, b = b, a+b
print ('END')
```

Program flow loop to while and check condition again

Control Flow

```
a, b = 0, 1
while b < 10:
    print (b)
    a, b = b, a+b
print ('END')
```

3rd time : b = 2
Enter loop
print b

Output

```
1
1
2
```

Control Flow

```
a, b = 0, 1
while b < 10:
    print (b)
    a, b = b, a+b
print ('END')
```

Using old values of a & b

3rd time : b = 2

a update to 2
b update to 2+1

Control Flow

```
a, b = 0, 1
while b < 10:
    print (b)
    a, b = b, a+b
print ('END')
```

4th time : b = 3

Enter loop

print b

Output

```
1
1
2
3
```

Control Flow

```
a, b = 0, 1
while b < 10:
    print (b)
    a, b = b, a+b
print ('END')
```

4th time : b = 3

a update to 3
b update to 3+2

Using old values of a & b

Control Flow

```
a, b = 0, 1
while b < 10:
    print (b)
    a, b = b, a+b
print ('END')
```

5th time : b = 5

Enter loop

print b

Output

```
1
1
2
3
5
```

Control Flow

```
a, b = 0, 1
while b < 10:
    print (b)
    a, b = b, a+b
print ('END')
```

Using old values of a & b

5th time : b = 5

a update to 5

b update to 5+3

Control Flow

```
a, b = 0, 1
while b < 10:
    print (b)
    a, b = b, a+b
print ('END')
```

6th time : b = 8
Enter loop
print b

Output

```
1
1
2
3
5
8
```

Control Flow

```
a, b = 0, 1
while b < 10:
    print (b)
    a, b = b, a+b
print ('END')
```

5th time : b = 8

a update to 8
b update to 8+3
(now 11)

Control Flow

```
a, b = 0, 1
while b < 10:
    print (b)
    a, b = b, a+b
print ('END')
```

Program flow exit loop as b< 10 is now false

Output

```
1
1
2
3
5
8
END
```

Control Flow

```
for x in range(0, 3) :
    print (x)
```

Output

```
0
1
2
```

More on Flow Control, and Output Formatting

Conditional Expressions

- In Python, there is a conditional expression construct similar to the ternary operator in C.
- Syntax:

```
var = true_value if condition else false_value
```

which is equivalent to:

```
if condition:
    var = true_value
else:
    var = false_value
```

- Example:

```
a, b = 10, 20
min = a if a < b else b
print(min)  # output 10
```

⟷

```
a, b = 10, 20
if a < b:
    min = a
else :
    min = b
print(min)  # output 10
```

Output Formatting using a string modulo

- Much like a printf()-style format as in C language, old formatting style from Python 2
- Examples:

```
print('%s %s' % ('one','two'))
```

```
one two
```

```
b = 2.345
print('%02d %.2f' % (1,b))
```

```
01 2.35
```

Output Formatting using the format method

- Examples:

```
print('{} {}'.format('one','two'))
```

```
one two
```

```
b = 2
print('{} {}'.format(1,b))
```

```
1 2
```

Output Formatting using the format method

- With new style formatting, it is possible (and in Python 2.6 even mandatory) to give placeholders an explicit positional index.
- This allows for rearranging the order of display without changing the arguments.

```
print('{1} {0}'.format('one','two'))
```

```
two one
```

This operation is not available with old-style formatting.

```
b = 2.345
print('{0} {1:5.2f}'.format(1,b))
```

```
1  2.35
```

Output Formatting using the format method

- Indeed, the placeholders can be identified by named indexes {price}, numbered indexes {0}, or empty braces {}.

```
txt = "For only {price:.2f} dollars!"
print(txt.format(price = 32.1))
```

```
For only 32.10 dollars!
```

```
txt1 = "My name is {fname}, I'am {age}".format(fname = "Noelle", age = 2.5)
txt2 = "My name is {0}, I'am {1}".format("Noelle", 2.5)
txt3 = "My name is {}, I'am {}".format("Noelle", 2.5)
print(txt1)
print(txt2)
print(txt3)
```

```
My name is Noelle, I'am 2.5
My name is Noelle, I'am 2.5
My name is Noelle, I'am 2.5
```