w01-Lec

Types, Literals, Variables, Operators, and Expressions

Assembled for 204217

2015 S1

by Kittipitch Kuptavanich

Values and Types

- ใน Python มีชนิดข้อมูลพื้นฐาน อยู่สองประเภทคือ
  - แบบที่ไม่สามารถแบ่งย่อยลงไปได้อีก (atomic, scalar) มี 4 ชนิดได้แก่
    - `int` – แทนจำนวนเต็ม เช่น 3, -8
    - `float` – แทนจำนวนจริง เช่น 2.36
    - `bool` – แทนค่าทางตรรกะ `True` (จริง) หรือ `False` (เท็จ)
    - `None` – เป็นชนิดข้อมูลที่มีค่าเป็น None ได้อย่างเดียว
  - แบบที่สามารถแบ่งย่อยลงไปได้ เช่น
    - `str` – สายอักขระ เช่น 'hello' "ก" หรือ "" (สังเกตเครื่องหมายคำพูด) สามารถเข้าถึงข้อมูลแยกทีละอักขระได้
    - `complex` – จำนวนเชิงซ้อน ประกอบด้วย ส่วน Real และส่วน Imaginary เช่น `1 + 2j`

Values and Types [2]

- ตัวเลข หรือ สายอักขระ (ที่อยู่ระหว่างเครื่องหมายคำพูด) ในโปรแกรมใด ๆ ถือเป็นค่าคงที่ (Literals)
- เราสามารถตรวจสอบชนิดของ Literals (และ Variable) ใน Python ได้โดยการใช้ ฟังก์ชัน `type()`

python shell

```
>>> type('Hello, World!')
<class 'str'>
>>> type(17)
<class _____>
>>> type(3.2)
<class _____>
>>> type('17')
<class _____>
>>> type(071)
<class _____>
```

Some Built-in Types

```
01 # Some Builtin Types
02 import math
...
05 def f():
06     print("This is a user-defined function")
07
08 print("Some basic types in Python:")
09 print(type(2))           # int
10 print(type(2 ** 500))    # Long int
11 print(type(2.2))         # float
12 print(type("2.2"))       # str  (string)
13 print(type(2 < 2.2))     # bool (boolean)
14 print(type(math))        # module
15 print(type(math.tan))    # builtin_function_or_method
16 print(type(f))           # function (user-defined function)
17 print(type(type(42)))    # type
```

Some Built-in Types [2]

```
23 print("Later in the course...")
24 print(type(Exception()))     # Exception
25 print(type(range(5)))        # range
26 print(type([1, 2, 3]))       # list
27 print(type((1, 2, 3)))       # tuple
28 print(type({1, 2}))          # set
29 print(type({1: 42}))         # dict (dictionary or map)
30 print(type(2 + 3j))          # complex  (complex number)
31
32 # Some Builtin Constants
33 print("Some builtin constants:")
34 print(True)
35 print(False)
36 print(None)
```

Variables

- **Variable (ตัวแปร) เป็นชื่อที่ใช้อ้างถึงข้อมูล (Data Object)**
- **การสร้าง Variable ใน Python ทำได้โดยการตั้งค่าให้กับชื่อ โดยการใช้เครื่องหมาย = (เท่ากับ)**

```
pi = 3.14
radius = 11
area = pi * radius * radius
```

- **ตัวแปร area มีชนิดข้อมูลเป็น_____?**
- **คำสั่งที่ใช้สร้าง Variable ขึ้นพร้อม ๆ กับให้ค่าในลักษณะนี้เรียกว่า Assignment Statement**

Aside

Expressions and Statements

- **An expression is a combination of values, variables, and operators.**
  - **An expression can be evaluated to a value**
    เราสามารถประเมินค่าของ Expression ได้
- **A statement is a unit of code that the Python interpreter can execute.**
  Statement คือหน่วยย่อยของชุดคำสั่งที่ Python Interpreter ดำเนินการได้
  - **For example, assignment statement**
- **Expression has value; a statement does not.**

```
>>> x = 3        # statement
>>> x == 3       # expression
True
>>> x            # expression
3
```

Variables [2]

```
>>> theSum = 0
>>> theSum
0
>>> theSum = theSum + 1
>>> theSum
1
>>> theSum = True
>>> theSum
True
```

ในภาษา Programming หลาย ๆ ภาษาเช่น Python ชนิดของตัวแปรสามารถเปลี่ยนแปลงได้ (Dynamic Typing)

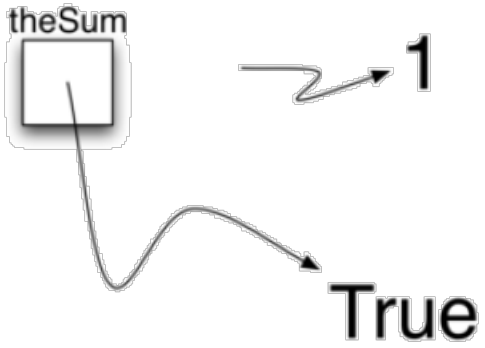


Figure 1.3: Variables Hold References to Data Objects

Figure 1.4: Assignment changes the Reference

Variable Names

- ความยาวไม่จำกัด
- ใช้ได้แค่ตัวอักษร ตัวเลข และเครื่องหมาย Underscore
- อักขระตัวแรกของชื่อ Variable ต้องเป็นตัวอักษรเท่านั้น (ควรใช้ตัวพิมพ์เล็ก)
- ชื่อ Variable (หรือ Identifier อื่น ๆ) นั้นมีความเป็น Case Sensitive กล่าวคือ `value` `Value` `vAlue` และ `valUE` ถือเป็น Variable คนละตัวกัน
- จะต้องไม่ซ้ำกับ Keyword ใน Python
- มาตรฐานการตั้งชื่อ Variable ใน Python ให้ใช้ตัวพิมพ์เล็กทั้งหมดและพิจารณาการใช้ Underscore คั่นระหว่างคำเพื่อทำให้อ่านง่ายขึ้น เช่น `max_score`

Aside

Python Keywords

```
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
```

- เราสามารถแสดง **list** ของ **keyword** ได้โดยการใช้คำสั่ง

```
>>> import keyword
>>> keyword.kwlist
```

Variable Names [2]

- พิจารณาชุดคำสั่งด้านล่าง

```
a = 3.14159                pi = 3.14159
b = 11.2          VS       diameter = 11.2
c = a * b * b              area = pi * diameter * diameter
```

- ในมุมมองของ **Python Interpreter** ทั้งสองคำสั่งมีความหมายเหมือนกัน
- แต่ในสายตาผู้อ่าน ชุดคำสั่งทางด้านซ้าย ดูเหมือนทำงานได้เป็นปกติ
- ในขณะที่ชุดคำสั่งทางด้านขวา อาจมีข้อผิดพลาด
  - ชื่อตัวแปรควรเป็น `radius` แทนที่จะเป็น `diameter`?
  - หรือควรนำ `diameter` มาหารด้วย 2 ก่อนนำไปหาพื้นที่?
- การตั้งชื่อตัวแปรที่ดี ช่วยทำให้ **Code** เข้าใจง่ายและลดข้อผิดพลาด

หมายเหตุ ในการตั้งชื่อตัวแปรที่ใช้เก็บค่าที่ได้จากการวัดที่มีหน่วยต่าง ๆ กัน ควรมีการระบุหน่วยในชื่อตัวแปร เพื่อความชัดเจน เช่น `len_km`, `speed_mph`, `weight_lb`

Numeric and Boolean Operators

Category	Operators
Arithmetic	+, -, *, /, //, **, %, - (unary), + (unary)
Relational	<, <=, >=, >, ==, !=
Bitwise	<<, >>, &, |, ^, ~
Assignment	+=, -=, *=, /=, //=, **=, %=, <<=, >>=, &=, |=, ^=
Logical	and, or, not

Note

- `/` is normal division — `3 / 2 == 1.5`
- `//` is *floored* division — `3 // 2 == 1`, `-3 // 2 == -2`
- `**` is power

Types Affect Semantics

```
>>> print(3 * 2)
6

>>> print(3 * "abc")
abcabcabc

>>> print(3 + 2)
5

>>> print("abc" + "def")
abcdef

>>> print(3 + "def")
TypeError: unsupported operand type(s) for +: 'int' and
'str'
```

Operator Precedence

- Operator ใน*ทางคณิตศาสตร์*ใน Python เป็นไปตามกฎการคำนวณปกติ (PEMDAS) โดยมีลำดับการดำเนินการดังนี้
  - Parentheses
  - Exponentiation
  - Multiplication and Division
  - Addition and Subtraction
- ในกรณีที่ operator อยู่ในลำดับเดียวกัน เช่น + และ – ให้ทำ Operation จากซ้ายไปขวา
- 2^{3^5} มีค่าเท่ากับเท่าไร

```
>>> 2**3**5 == (2**3)**5

>>> 2**3**5 == 2**(3**5)
```

Operator Precedence [2]

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

Mathematical Operators

high

low

Boolean Expressions

- **Boolean Expression คือ Expression ที่มีค่าเป็น `True` (จริง) หรือ `False` (เท็จ)**

```
>>> 5 == 5
True
>>> 5 == 6
False
```

- **ค่า `True` หรือ `False` เป็นค่าเฉพาะที่มาจากชนิดข้อมูล `bool` (และไม่ใช่ string)**

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

http://www.cs.cmu.edu/~112/notes/notes-data-and-exprs.html

Boolean Expressions [2]

- เครื่องหมาย == เป็นหนึ่งใน Operator ทางความสัมพันธ์ (Relational Operator)
  - Relational Operator อื่น ๆ ได้แก่

```
x != y      # x is not equal to y
x > y       # x is greater than y
x < y       # x is less than y
x >= y      # x is greater than or equal to y
x <= y      # x is less than or equal to y
```

- หากต้องการเขียน Expression ข้ามบรรทัด สามารถทำได้โดยการใช้เครื่องหมาย Backslash \ หรือวงเล็บ ( )

```
>>> x = 8
>>> (x + 4 < 10 and
     x % 2 != 1)
False
```

```
>>> x = 8
>>> x + 4 < 10 and \
    x % 2 != 1
False
```

http://www.cs.cmu.edu/~112/notes/notes-data-and-exprs.html

Boolean Expressions [2]

- **Floating-point Number Comparisons**

```
>>> print(0.1 + 0.1 == 0.2)
True                  # True, but...

>>> print(0.1 + 0.1 + 0.1 == 0.3)
False

>>> print(0.1 + 0.1 + 0.1)
0.3                   # seems ok

>>> print((0.1 + 0.1 + 0.1) - 0.3)
5.55111512313e-17     # (tiny, but non-zero!)
```

- ค่าที่เก็บในตัวแปรชนิด `float` เป็นค่าประมาณ!!

http://www.cs.cmu.edu/~112/notes/notes-data-and-exprs.html

Floating-Point Numbers and `almost_equal()`

```
09 d1 = 0.1 + 0.1 + 0.1
10 d2 = 0.3
11 print(d1 == d2)                  # still False, of course
12 epsilon = 10 ** -10
13 print(abs(d2 - d1) < epsilon)  # True!
14
15 # Once again, using an almostEqual function
16 # (that we will write)
17
18 def almostEqual(d1, d2, epsilon=10 ** -10):
19     return (abs(d2 - d1) < epsilon)
20 d1 = 0.1 + 0.1 + 0.1
21 d2 = 0.3
22 print(d1 == d2)               # still False, of course
23 # True, and now packaged in a handy reusable function!
24 print(almostEqual(d1, d2))
```

Logical Operator

- ในภาษา Python มี ตัวดำเนินการทางตรรกะ (Logical Operator) 3 ตัวได้แก่ `and`, `or` และ `not`
- ความหมายของ Operator ทั้งสามตัวตรงกับความหมายในภาษาอังกฤษ
- ตัวอย่างเช่น

เราสามารถใช้วงเล็บเพื่อช่วยทำให้เงื่อนไขอ่านง่ายขึ้น

- `(x > 0) and (x < 10)`
- `(n % 2 == 0) or (n % 3 == 0)`

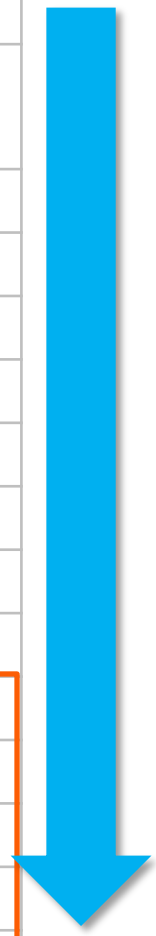
  - `x > 0 and x < 10` จะเป็นจริงก็ต่อเมื่อ `x` มากกว่า `0` และ น้อยกว่า `10` (ในกรณีนี้สามารถเขียน `0 < x < 10` ได้)
  - `n % 2 == 0 or n % 3 == 0` จะเป็นจริงเมื่อ เงื่อนไขตัวใดตัวหนึ่ง (หรือทั้ง 2 ตัว) เป็นจริง
  - โดยทั่วไปแล้ว Operands ของ Logical Operator ควรเป็น Boolean Expression แต่ในภาษา Python Truth Value ของตัวเลขใด ๆ ที่มีค่าไม่เป็น 0 จะมีค่าเป็น `True`

```
>>> 17 and True
True
```

Operator Precedence [2]

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

high



low

Short-Circuit Evaluation

- หากพิจารณา Truth Table ของ Expression `p or q`
- จะพบว่า กรณีเดียวที่ Expression จะมีค่าเป็น `False` คือกรณีที่ `p` และ `q` เป็น `False` ทั้งคู่
  - ดังนั้นหาก หากพบว่า Operand ตัวแรก (`p`) มีค่าเป็น `True` เราสรุปได้ว่า Expression นี้จะ Evaluate เป็น `True` โดยไม่จำเป็นต้องพิจารณาค่าของ `q`
  - กรณี `p` มีค่าเป็น `False` เป็นกรณีเดียวที่ต้องพิจารณาค่า `q`
- การ Evaluate ค่าโดยพิจารณา Operand บางส่วนเท่าที่จำเป็นแล้วให้ผลลัพธ์ทันที เรียกว่า **Short Circuit Evaluation**
- ภาษา Programming หลาย ๆ ภาษา ใช้การ Evaluate ในลักษณะนี้
- เช่นเดียวกันกับการพิจารณา Expression `p and q`
  - ทำ Short Circuit Evaluation ได้ทันทีเมื่อพบว่า `p` มีค่าเป็น_____ และจะ Evaluate Expression เป็น_____

p	*q*	*p* ∨ *q*
T	T	T
T	F	T
F	T	T
F	F	F

p	*q*	*p* ∧ *q*
T	T	T
T	F	F
F	T	F
F	F	F

http://www.cs.cmu.edu/~112/notes/notes-data-and-exprs.html

Short-Circuit Evaluation [2]

```
>>> x = 1
>>> y = 0
>>> print((y != 0) and ((x / y) != 0))  # Works!
False
>>> print(((x / y) != 0) and (y != 0))  # Crashes!
...
ZeroDivisionError: division by zero

>>> print((y == 0) or ((x / y) == 0))   # Works!
True
>>> print(((x / y) == 0) or (y == 0))   # Crashes!
...
ZeroDivisionError: division by zero
```

http://www.cs.cmu.edu/~112/notes/notes-data-and-exprs.html

Short-Circuit Evaluation [3]

```
25 def isPositive(n):
26     result = (n > 0)
27     print("isPositive(", n, ") =", result)
28     return result
29
30
31 def isEven(n):
32     result = (n % 2 == 0)
33     print("isEven(", n, ") =", result)
34     return result
35
36 print("Test 1: isEven(-4) and isPositive(-4)")
37 print(isEven(-4) and isPositive(-4))  # Calls both
38 print("----------")
39 print("Test 2: isEven(-3) and isPositive(-3)")
40 print(isEven(-3) and isPositive(-3))  # Calls only one
```

Truth Value Testing

- ในภาษา Python, Object ทุกตัวมี Truth Value ทั้งหมด (สามารถ Evaluate เป็น `True` หรือ `False` ได้)
- Value ดังต่อไปนี้ Evaluate เป็น `False` (ที่เหลือเป็น `True`)
  - `None`
  - `False`
  - zero of any numeric type, for example, `0, 0.0, 0j.`
  - any empty sequence, for example, `'', (), [].`
  - any empty mapping, for example, `{}`.
  - instances of user-defined classes, if the class defines a `__bool__()` or `__len__()` method, when that method returns the integer `zero` or bool value `False`.

Truth Value Testing [2]

- **Operation และฟังก์ชัน Built-in ใด ๆ ที่มีการคืนค่าเป็น Boolean จะคืนค่าดังต่อไปนี้เท่านั้น**
  - **1 หรือ True ถ้าเป็นจริง และ**
  - **0 หรือ False ถ้าเป็นเท็จ เสมอ**
- ***ข้อยกเว้น*** **Operator and และ or จะคืนค่าเป็น Operand ตัวใดตัวหนึ่ง (พิจารณาแบบ Short Circuit Evaluation)**

```
>>> 23 and 35
35
>>> 0 and -23
0
>>> True and 5
5
```

```
>>> False and 5
False
>>> (1 and 2) or 42
2
>>> (1 and 0) or 42
42
```

Boolean Arithmetic

```
02 # In numeric expressions...
03 #     True is treated as 1
04 #     False is treated as 0
05
06 # So...
07 print(5 * True)  # 5
08 print(5 * False) # 0
09 print(5 + True)  # 6
10 print(5 + False) # 5
```

- ไม่ควรใช้ **Boolean Arithmetic** เนื่องจากทำให้ **Code** อ่านยาก แต่หากจำเป็นควรมีการ **Cast** ชนิด ด้วยฟังก์ชัน `int()` ก่อน

```
07 print(5 * int(True))  # 5
08 print(5 * int(False)) # 0
09 print(5 + int(True))  # 6
10 print(5 + int(False)) # 5
```

Bitwise Operations

ตัวอย่าง

- ให้ a = [0110] และ b = [1100]
- ในการทำ **bitwise operation** a & b, a | b, a ^ b, และ ~b

จะได้ผลลัพธ์ดังนี้

```
  0110        0110        0110        ~ 1100
& 1100      | 1100      ^ 1100        ------
------      ------      ------        0011
0100        1110        1010
```

Bitwise Operations [2]

```
# 0 1 1 0 = 4 + 2 = 6
# 0 1 0 1 = 4 + 1 = 5
>>> print("6 & 5 =", (6 & 5))
6 & 5 = _____
>>> print("6 | 5 =", (6 | 5))
6 | 5 = _____
>>> print("6 ^ 5 =", (6 ^ 5))
6 ^ 5 = _____
>>> print("6 << 1 =", (6 << 1))
6 << 1 = _____
>>> print("6 << 2 =", (6 << 2))
6 << 2 = _____
>>> print("6 >> 1 =", (6 >> 1))
6 >> 1 = _____
```

Python bitwise ~ Operator

- Python ใช้การแทนค่าตัวเลขแบบ two's complement
  - จำนวนเต็มบวกใด ๆ มีค่า MST เป็น 0
  - จำนวนเต็มลบใด ๆ มีค่า MST เป็น 1
- พิจารณาค่า 3 (11) ในภาษา Python
  - การแทนค่าจึงอยู่ในรูป [011] เพิ่ม MST = 0
- ดังนั้น ~3 ใน Python จึงมีค่า [100]
  - ตีความแบบ two's complement ได้เป็น -4

```
>>> ~3
-4
```

Python bitwise ~ Operator [2]

- เช่นเดียวกันในกรณีจำนวนลบ เช่น -35
  - เขียนแบบ two's complement
    - 35 = [0100011] ดังนั้น
    - -35 = [1011101] กรณีนี้ไม่ต้องเพิ่ม MST
  - ~-35 จึงมีค่า [0100010] = 34

```
>>> ~-35
34
>>> int('0100010', 2)
34
```

- ค่า ~x จะมีค่าเท่ากับ -x - 1 ทั้งจำนวน + - และ 0

- สังเกตการใช้ฟังก์ชัน `int()` เพื่อแปลงเลขฐาน 2 เป็น integer
- เลข 2 เป็น optional parameter (default คือ ฐาน 10)

Python Script Mode

- ที่ bash prompt สร้าง file เปล่า (คำสั่ง touch) แล้วเปิดไฟล์มา edit ด้วย IDLE

bash shell
```
$ touch hello.py
$ idle hello.py &
```

- Python script ควรมีการระบุบรรทัดแรกเป็น `#!/usr/bin/env python3` (ไม่ต้องพิมพ์เลขบรรทัด) เพื่อระบุว่าเป็น Script ของ Python 3

hello.py
```
01 #!/usr/bin/env python3
02
```

- จากนั้นให้แสดง String `'Hello World!!'` โดยใช้ฟังก์ชัน `print()`

hello.py
```
01 #!/usr/bin/env python3
02
03 print("Hello World!!")
```

Python Script Mode [2]

- **Save แล้ว กด F5 ในหน้าต่าง IDLE เพื่อ run script**

```
>>>
Hello World!!
>>>
```

- **เราสามารถ run script จากหน้าต่าง bash shell ได้เช่นกัน**

```
$ python hello.py
Hello World!!
```

- **หรือเปลี่ยนประเภทไฟล์ให้เป็น executable ด้วยคำสั่ง** `chmod +x` **(ทำครั้งเดียว) แล้ว run ด้วยชื่อไฟล์ (ต้องใส่ path ในที่นี้คือ** `./`**)**

```
$ chmod +x hello.py
$ ./hello.py
Hello World!!
```

Cygwin, Linux and OS X only

Python Script Mode [3]

```
>>> x = 'hello'
>>> y = 'world'
>>> x + y
'helloworld'
```

- ใน **Interactive Mode** เมื่อพิมพ์ **Expression** ใด ๆ ลงไป **Python Shell** จะแสดงค่าของ **Expression** นั้น ๆ

hello_world.py

```
08 x = 'hello'
09 y = 'world'
10 x + y
```

- แต่หากเรา **Run Script** ด้านบนจะพบว่าไม่มี **Output** ใด ๆ

```
$ python hello_world.py
```

- ใน **Script mode** หากต้องการให้มีการแสดงผล **Expression** ใด ๆ เราจำเป็นต้องใช้ฟังก์ชัน `print()`

```
10 print(x + y)
```

References

- **https://docs.python.org/3/library/stdtypes.html**
- **https://docs. Python.org/3.4/reference/expressions.html**
- **https://docs. Python.org/3.4/tutorial/inputoutput.html**
- **https://docs. Python.org/3.4/library/stdtypes.html#old-string-formatting**
- **http://www.cs.cmu.edu/~112/notes/notes-data-and-exprs.html**
- **Miller, B., and Ranum, D. *Problem Solving with Algorithms and Data Structures Using Python*,**
- **Guttag, John V. *Introduction to Computation and Programming Using Python, Revised***