

LẬP TRÌNH VỚI PYTHON

(Programming with Python)

Th.S Nguyễn Hoàng Thành

Email: thanhnh@ptithcm.edu.vn

Tel: 0909 682 711

Numeric Types

1 Numeric Type Basics

- **Python** provides more **advanced numeric programming support** and **objects** for more advanced work. A complete inventory of Python's numeric toolbox includes:
 - Integers and floating-point numbers
 - Complex numbers
 - Fixed-precision decimal numbers
 - Rational fraction numbers
 - Sets
 - Booleans
 - Unlimited integer precision
 - A variety of numeric built-ins and modules

1.1 Numeric Literals

- Offers a complex number type
- Python provides
 - integers (positive and negative whole numbers) and
 - floating-point numbers
- Python allows
 - us to **write integers** using hexadecimal, octal and binary literals;
 - integers to have unlimited precision

Table 5-1. Basic numeric literals

Literal	Interpretation
1234, -24, 0, 9999999999999999	Integers (unlimited size)
1.23, 1., 3.14e-10, 4E210, 4.0e+210	Floating-point numbers
0177, 0x9ff, 0b101010	Octal, hex, and binary literals in 2.6
0o177, 0x9ff, 0b101010	Octal, hex, and binary literals in 3.0
3+4j, 3.0+4.0j, 3J	Complex number literals

1.2 Built-in Numeric Tools

- Python provides a set of tools for processing number objects:

- **Expression** operators

$+$, $-$, $*$, $/$, $>>$, $**$, $\&$, etc.

- Built-in **mathematical** functions

`pow`, `abs`, `round`, `int`, `hex`, `bin`, etc.

- **Utility modules**

`random`, `math`, etc.

1.3 Python Expression Operators

Operators	Description
yield x	Generator function send protocol
lambda args: expression	Anonymous function generation
x if y else z	Ternary selection (x is evaluated only if y is true)
x or y	Logical OR (y is evaluated only if x is false)
x and y	Logical AND (y is evaluated only if x is true)
not x	Logical negation
x in y, x not in y	Membership (iterables, sets)

1.3 Python Expression Operators (con't)

Operators	Description
<code>x is y, x is not y</code>	Object identity tests
<code>x < y, x <= y, x > y, x >= y</code> <code>x == y, x != y</code>	Magnitude comparison, set subset and superset; Value equality operators
<code>x y</code>	Bitwise OR, set union
<code>x ^ y</code>	Bitwise XOR, set symmetric difference
<code>x & y</code>	Bitwise AND, set intersection
<code>x << y, x >> y</code>	Shift x left or right by y bits
<code>x + y</code> <code>x - y</code>	Addition, concatenation; Subtraction, set difference

1.3 Python Expression Operators (con't)

Operators	Description
$x * y$ $x \% y$ $x / y, x // y$	Multiplication, repetition; Remainder, format; Division: true and floor
$-x, +x$	Negation, identity
$\sim x$	Bitwise NOT (inversion)
$x ** y$	Power (exponentiation)
$x[i]$	Indexing (sequence, mapping, others)
$x[i:j:k]$	Slicing
$x(...)$	Call (function, method, class, other callable)

1.3 Python Expression Operators (con't)

Operators	Description
x.attr	Attribute reference
(...)	Tuple, expression, generator expression
[...]	List, list comprehension
{...}	Dictionary, set, set and dictionary comprehensions

Mixed operators follow operator precedence

- When you write an expression with more than one operator, Python **groups** its parts according to what are called **precedence rules**, and this **grouping determines the order** in which the expression's parts are computed
 - Operators **lower** in the table have **higher precedence**,
 - Generally group from **left to right** when combined

$$X + Y * Z$$

$$(X + Y) * Z$$

$$X + (Y * Z)$$

2. Numbers in Action

- Best way to understand numeric objects and expressions is
 - to see them in action,
 - start up the interactive command line and
 - try some basic but illustrative operations

2.1 Variables and Basic Expressions

- First assign two variables (a and b) to integers.
- Variables
 - simply **names** created by you or Python .
 - **created** when they are **first assigned** values.
 - **replaced** with **their values** when used in expressions.
 - must be **assigned** before they can be used in expressions.
 - refer to objects and
 - **never declared** ahead of time.

2.1 Variables and Basic Expressions (con't)

```
Python 3.7 (64-bit)
>>>
>>> a = 3 #Name created
>>> b = 4
>>> a + 1, a - 1
(4, 2)
>>> b * 3, b / 2
(12, 2.0)
>>> a % 2, b ** 2
(1, 16)
>>> 2 + 4.0, 2.0 ** b
(6.0, 16.0)
>>>
```

**Name created
with comment**

2.1 Variables and Basic Expressions (con't)

```
Python 3.7 (64-bit)
>>>
>>> a = 3   #Name created
>>> b = 4
>>> a + 1, a - 1
(4, 2)
>>> b * 3, b / 2
(12, 2.0)
>>> a % 2, b ** 2
(1, 16)
>>> 2 + 4.0, 2.0 ** b
(6.0, 16.0)
>>>
```

**Addition ($3 + 1$),
subtraction ($3 - 1$)**

2.1 Variables and Basic Expressions (con't)

```
Python 3.7 (64-bit)
>>>
>>> a = 3  #Name created
>>> b = 4
>>> a + 1, a - 1
(4, 2)
>>> b * 3, b / 2
(12, 2.0)
>>> a % 2, b ** 2
(1, 16)
>>> 2 + 4.0, 2.0 ** b
(6.0, 16.0)
>>>
```

Multiplication ($4 * 3$), division ($4 / 2$)

2.1 Variables and Basic Expressions (con't)

```
Python 3.7 (64-bit)
>>>
>>> a = 3  #Name created
>>> b = 4
>>> a + 1, a - 1
(4, 2)
>>> b * 3, b / 2
(12, 2.0)
>>> a % 2, b ** 2
(1, 16)
>>> 2 + 4.0, 2.0 ** b
(6.0, 16.0)
>>>
```

**Modulus (remainder),
power ($4 ** 2$)**

2.1 Variables and Basic Expressions (con't)

```
Python 3.7 (64-bit)
>>>
>>> a = 3   #Name created
>>> b = 4
>>> a + 1, a - 1
(4, 2)
>>> b * 3, b / 2
(12, 2.0)
>>> a % 2, b ** 2
(1, 16)
>>> 2 + 4.0, 2.0 ** b
(6.0, 16.0)
>>>
```

**Mixed-type
conversions**

2.2 Numeric Display Formats

```
Python 3.7 (64-bit)
>>>
>>> num = 1 / 3.0
>>> num
0.3333333333333333
>>> '%e' % num
'3.333333e-01'
>>> '%4.2f' % num
'0.33'
>>> '{0:4.2f}'.format(num)
'0.33'
>>>
```

Echoes

2.2 Numeric Display Formats

```
Python 3.7 (64-bit)
>>>
>>> num = 1 / 3.0
>>> num
0.3333333333333333
>>> '%e' % num
'3.333333e-01'
>>> '%4.2f' % num
'0.33'
>>> '{0:4.2f}'.format(num)
'0.33'
>>>
```

String format
expression

2.2 Numeric Display Formats

```
Python 3.7 (64-bit)
>>>
>>> num = 1 / 3.0
>>> num
0.3333333333333333
>>> '%e' % num
'3.333333e-01'
>>> '%4.2f' % num
'0.33'
>>> '{0:4.2f}'.format(num)
'0.33'
>>>
```

Alternative
Floating point
format

2.2 Numeric Display Formats

```
Python 3.7 (64-bit)
>>>
>>> num = 1 / 3.0
>>> num
0.3333333333333333
>>> '%e' % num
'3.333333e-01'
>>> '%4.2f' % num
'0.33'
>>> '{0:4.2f}'.format(num)
'0.33'
>>>
```

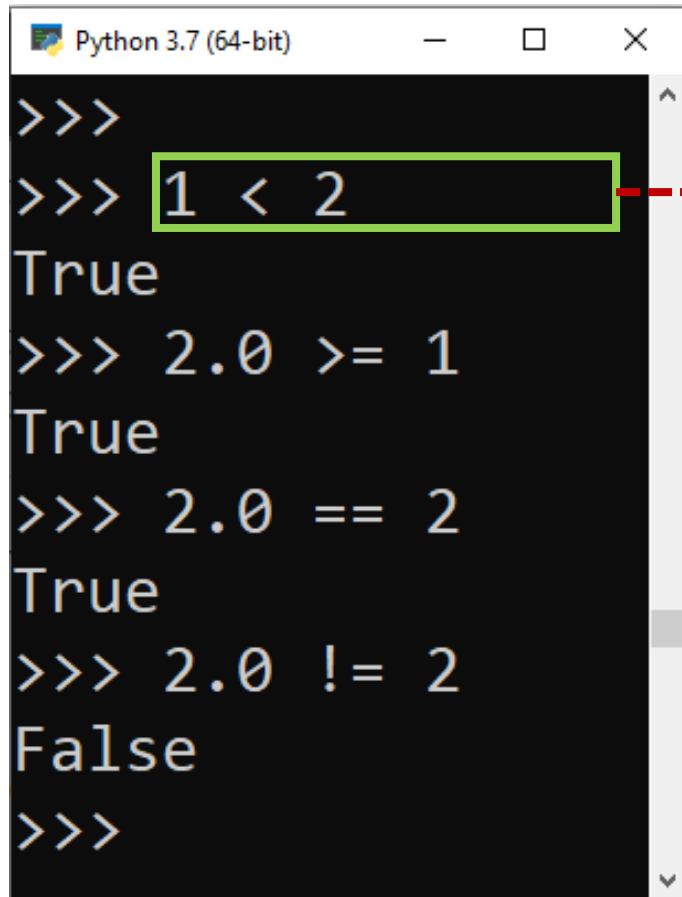
**String formatting
method**

2.3 Comparisons: Normal and Chained

- Number can also be compared
- compare the relative magnitudes of their operands and return a Boolean result

2.3 Comparisons: Normal and Chained (con't)

- **Normal Comparison**



```
>>>  
>>> 1 < 2  
True  
>>> 2.0 >= 1  
True  
>>> 2.0 == 2  
True  
>>> 2.0 != 2  
False  
>>>
```

Less than

2.3 Comparisons: Normal and Chained (con't)

- **Normal Comparison**

```
Python 3.7 (64-bit)
>>>
>>> 1 < 2
True
>>> 2.0 >= 1
True
>>> 2.0 == 2
True
>>> 2.0 != 2
False
>>>
```

Greater than or equal: mixed-type 1 converted to 1.0

2.3 Comparisons: Normal and Chained (con't)

- **Normal Comparison**

```
Python 3.7 (64-bit)
>>>
>>> 1 < 2
True
>>> 2.0 >= 1
True
>>> 2.0 == 2
True
>>> 2.0 != 2
False
>>>
```

Equal value



2.3 Comparisons: Normal and Chained (con't)

- **Normal Comparison**

```
Python 3.7 (64-bit)
>>>
>>> 1 < 2
True
>>> 2.0 >= 1
True
>>> 2.0 == 2
True
>>> 2.0 != 2
False
>>>
```

Not Equal value

2.3 Comparisons: Normal and Chained (con't)

- **Chained comparisons** are a sort of shorthand for larger Boolean expressions.
- The expression $(A < B < C)$, for instance,
 - tests whether B is between A and C;
 - it is equivalent to the Boolean test $(A < B \text{ and } B < C)$ but is easier on the eyes (and the keyboard).

2.3 Comparisons: Normal and Chained (con't)

- **Chained comparisons**

```
Python 3.7 (64-bit)
>>>
>>> X = 2
>>> Y = 4
>>> Z = 6
>>> X < Y < Z
True
>>> X < Y and Y < Z
True
>>>
```

```
Python 3.7 (64-bit)
>>>
>>> X = 2
>>> Y = 4
>>> Z = 6
>>> X < Y > Z
False
>>> X < Y and Y > Z
False
>>>
```

2.3 Comparisons: Normal and Chained (con't)

- **Chained comparisons**

```
Python 3.7 (64-bit)
>>>
>>> 1 < 2 < 3.0 < 4
True
>>> 1 > 2 > 3.0 > 4
False
>>> 1 == 2 < 3
False
>>> (1+1) == 2 < 3
True
>>>
```

(1 < 2) and (2 < 3.0) and (3.0 < 4)

2.3 Comparisons: Normal and Chained (con't)

- Chained comparisons

```
Python 3.7 (64-bit)
>>>
>>> 1 < 2 < 3.0 < 4
True
>>> 1 > 2 > 3.0 > 4
False
>>> 1 == 2 < 3
False
>>> (1+1) == 2 < 3
True
>>>
```

(1 > 2) and (2 > 3.0) and (3.0 > 4)

2.3 Comparisons: Normal and Chained (con't)

- Chained comparisons

```
Python 3.7 (64-bit)
>>>
>>> 1 < 2 < 3.0 < 4
True
>>> 1 > 2 > 3.0 > 4
False
>>> 1 == 2 < 3
False
>>> (1+1) == 2 < 3
True
>>>
```

(1 == 2) and (2 < 3)

2.3 Comparisons: Normal and Chained (con't)

- Chained comparisons

```
Python 3.7 (64-bit)
>>>
>>> 1 < 2 < 3.0 < 4
True
>>> 1 > 2 > 3.0 > 4
False
>>> 1 == 2 < 3
False
>>> (1+1) == 2 < 3
True
>>>
```

$((1+1) == 2) \text{ and } (2 < 3)$

Example:

```
>>> X = 2
```

```
>>> Y = 4
```

```
>>> Z = 6
```

The following two expressions have identical effects, but the first is shorter to type, and it may run slightly faster since Python needs to evaluate Y only once:

```
>>> X < Y < Z           # Chained comparisons: range tests
```

```
>>> X < Y and Y < Z
```

```
>>> X = 2
```

```
>>> Y = 4
```

```
>>> Z = 6
```

```
>>> X < Y > Z
```

```
#?
```

```
>>> X < Y and Y > Z
```

```
#?
```

```
>>> 1 < 2 < 3.0 < 4
```

```
#?
```

```
>>> 1 > 2 > 3.0 > 4
```

```
#?
```

2.4 Division: Classic, Floor, and True

- There is a difference between the division between Python versions 2.6 and Python 3.0
 - X / Y : Classic and true division.
 - 2.6: **truncating** results for integers, and keeping remainders
 - 3.0: always **keeping remainders** in floating-point results
 - $X // Y$: Floor division, it **truncates fractional remainders down to their floor**

2.4 Division: Classic, Floor, and True (con't)

Keep remainder

```
>>>  
>>> 10 / 4  
2  
>>> 10 // 4  
2  
>>> 10 / 4.0  
2.5  
>>> 10 // 4.0  
2.0  
>>>
```

Python version 2.6

#

```
>>>  
>>> 10 / 4  
2.5  
>>> 10 // 4  
2  
>>> 10 / 4.0  
2.5  
>>> 10 // 4.0  
2.0  
>>>
```

Python version 3.7

2.5 Integer Precision

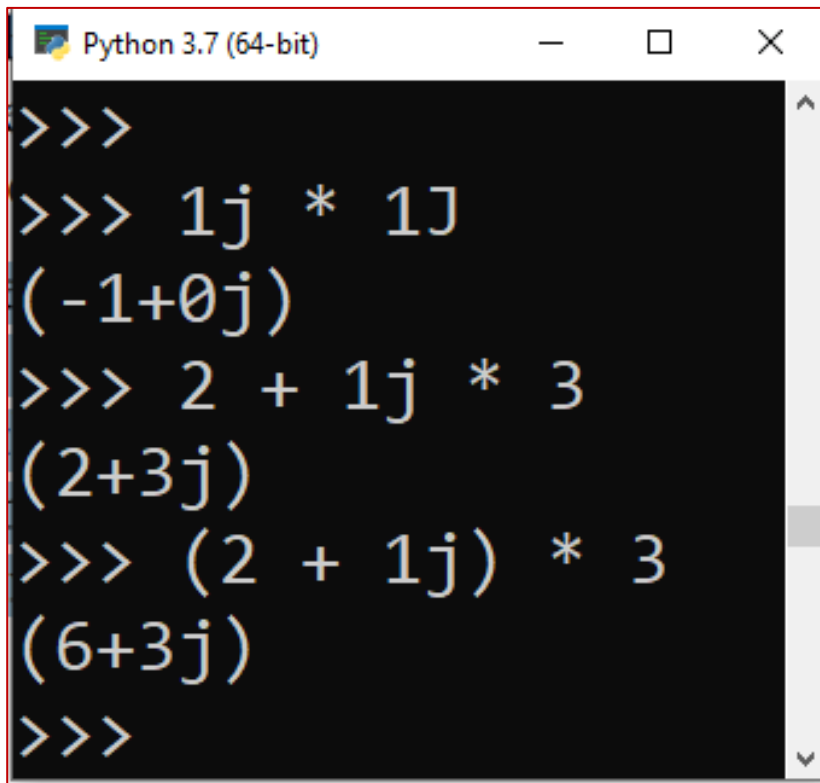
```
C:\Python26\python.exe
```

```
>>>  
>>> 999999999999999999999999999999999999 + 1  
1000000000000000000000000000000000000L  
>>> 2 ** 200  
16069380442589902755419620923411626025  
22202993782792835301376L  
>>>
```

[illegible]

2.6 Complex Numbers

- Complex numbers are represented as two floating-point numbers: the **real** and **imaginary** parts.
- coded by **adding a *j* or *J* suffix** to the **imaginary part**

A screenshot of a Python 3.7 (64-bit) shell window. The window has a title bar with the Python logo and text 'Python 3.7 (64-bit)', and standard window controls (minimize, maximize, close). The background is black with yellow text. The shell shows the following interactions:

```
>>>  
>>> 1j * 1j  
(-1+0j)  
>>> 2 + 1j * 3  
(2+3j)  
>>> (2 + 1j) * 3  
(6+3j)  
>>>
```

2.7 Hexadecimal, Octal, and Binary Notation

- Python integers can be coded in **hexadecimal**, **octal**, and **binary** notation.

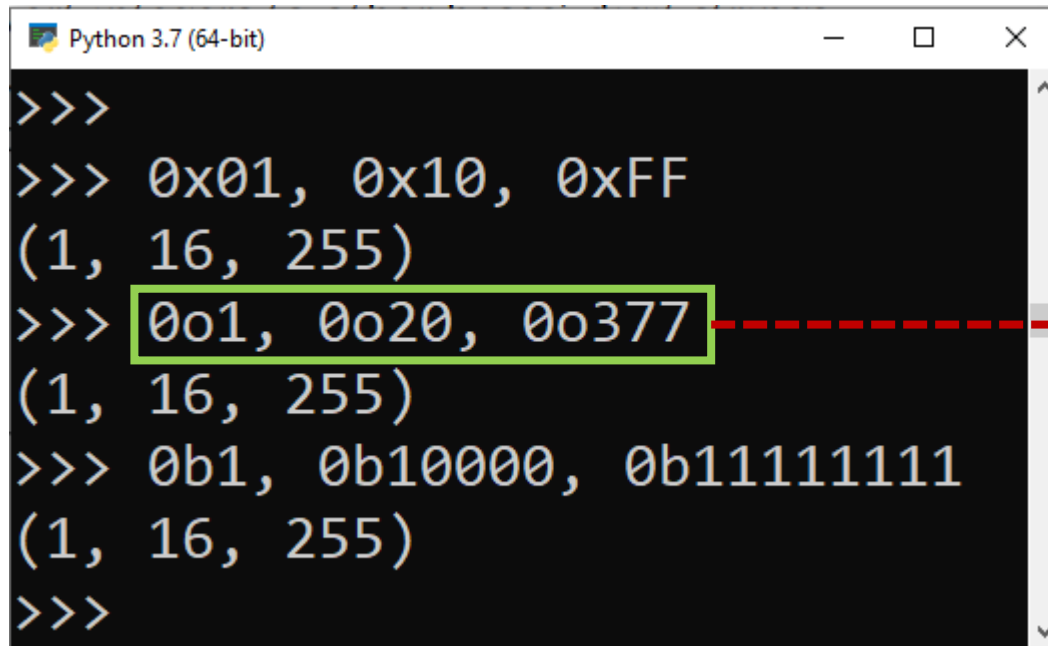
```
Python 3.7 (64-bit)
>>>
>>> 0x01, 0x10, 0xFF
(1, 16, 255)
>>> 0o1, 0o20, 0o377
(1, 16, 255)
>>> 0b1, 0b10000, 0b11111111
(1, 16, 255)
>>>
```

Hex literals

0x

2.7 Hexadecimal, Octal, and Binary Notation

- Python integers can be coded in **hexadecimal**, **octal**, and **binary** notation.



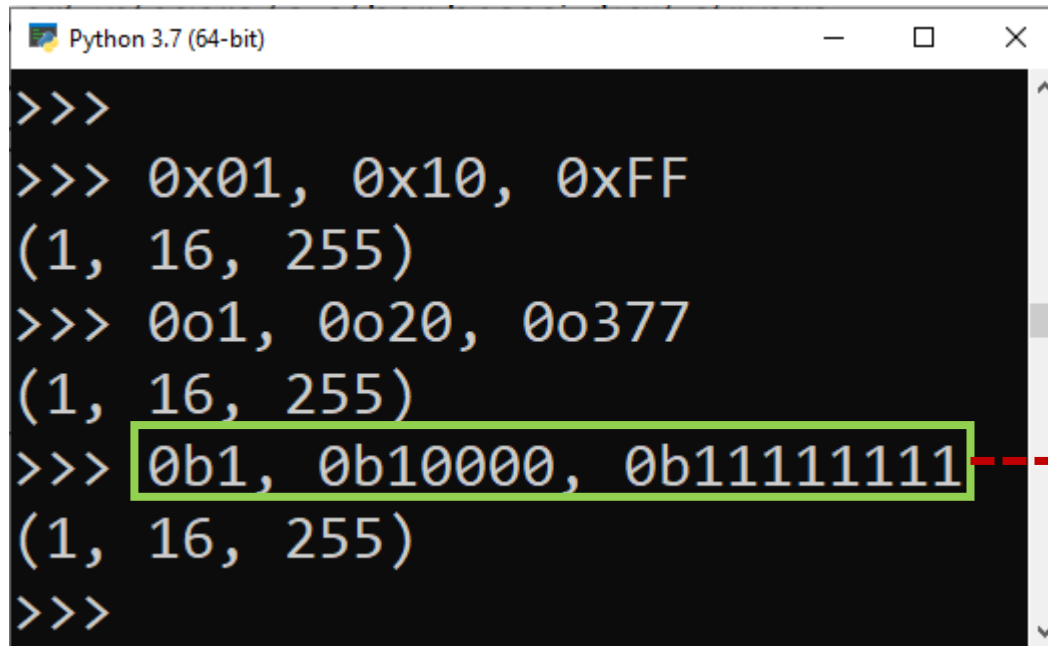
```
Python 3.7 (64-bit)
>>>
>>> 0x01, 0x10, 0xFF
(1, 16, 255)
>>> 0o1, 0o20, 0o377
(1, 16, 255)
>>> 0b1, 0b10000, 0b11111111
(1, 16, 255)
>>>
```

Octal literals

0o

2.7 Hexadecimal, Octal, and Binary Notation

- Python integers can be coded in **hexadecimal**, **octal**, and **binary** notation.



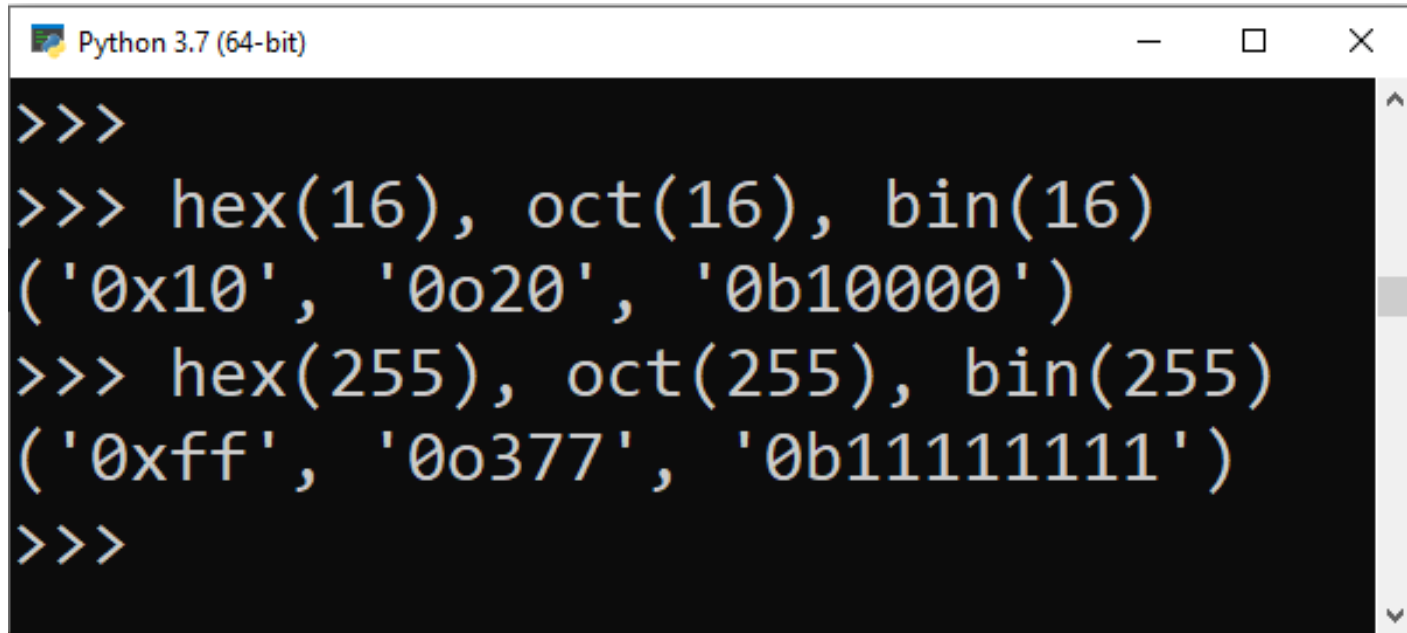
```
Python 3.7 (64-bit)
>>>
>>> 0x01, 0x10, 0xFF
(1, 16, 255)
>>> 0o1, 0o20, 0o377
(1, 16, 255)
>>> 0b1, 0b10000, 0b11111111
(1, 16, 255)
>>>
```

Binary literals

0b

2.7 Hexadecimal, Octal, and Binary Notation (con't)

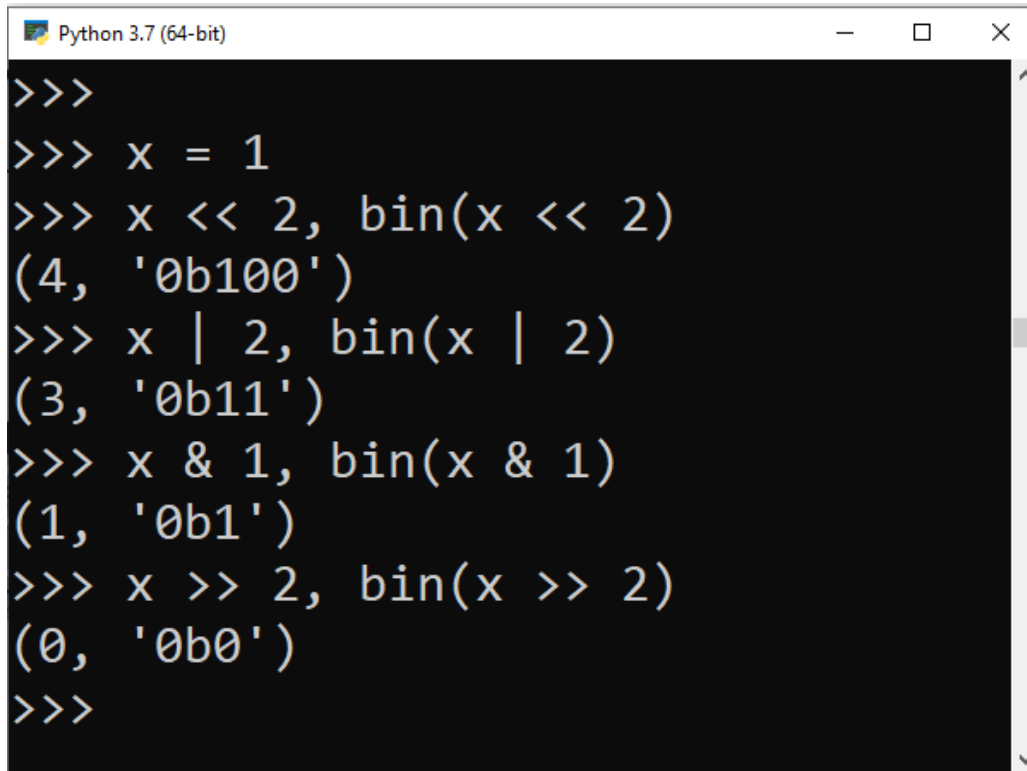
- Integer provides built-in functions -that allow you to convert **integers to other bases' digit strings**

A screenshot of a Python 3.7 (64-bit) terminal window. The window has a title bar with the Python logo and text "Python 3.7 (64-bit)", and standard window controls (minimize, maximize, close). The terminal background is black with yellow text. It shows a series of commands and their outputs: three prompt characters ">>>" followed by the command "hex(16), oct(16), bin(16)" which outputs "('0x10', '0o20', '0b10000')", then the command "hex(255), oct(255), bin(255)" which outputs "('0xff', '0o377', '0b11111111')", and finally another set of three prompt characters ">>>".

```
Python 3.7 (64-bit)
>>>
>>> hex(16), oct(16), bin(16)
('0x10', '0o20', '0b10000')
>>> hex(255), oct(255), bin(255)
('0xff', '0o377', '0b11111111')
>>>
```

2.8 Bitwise Operations

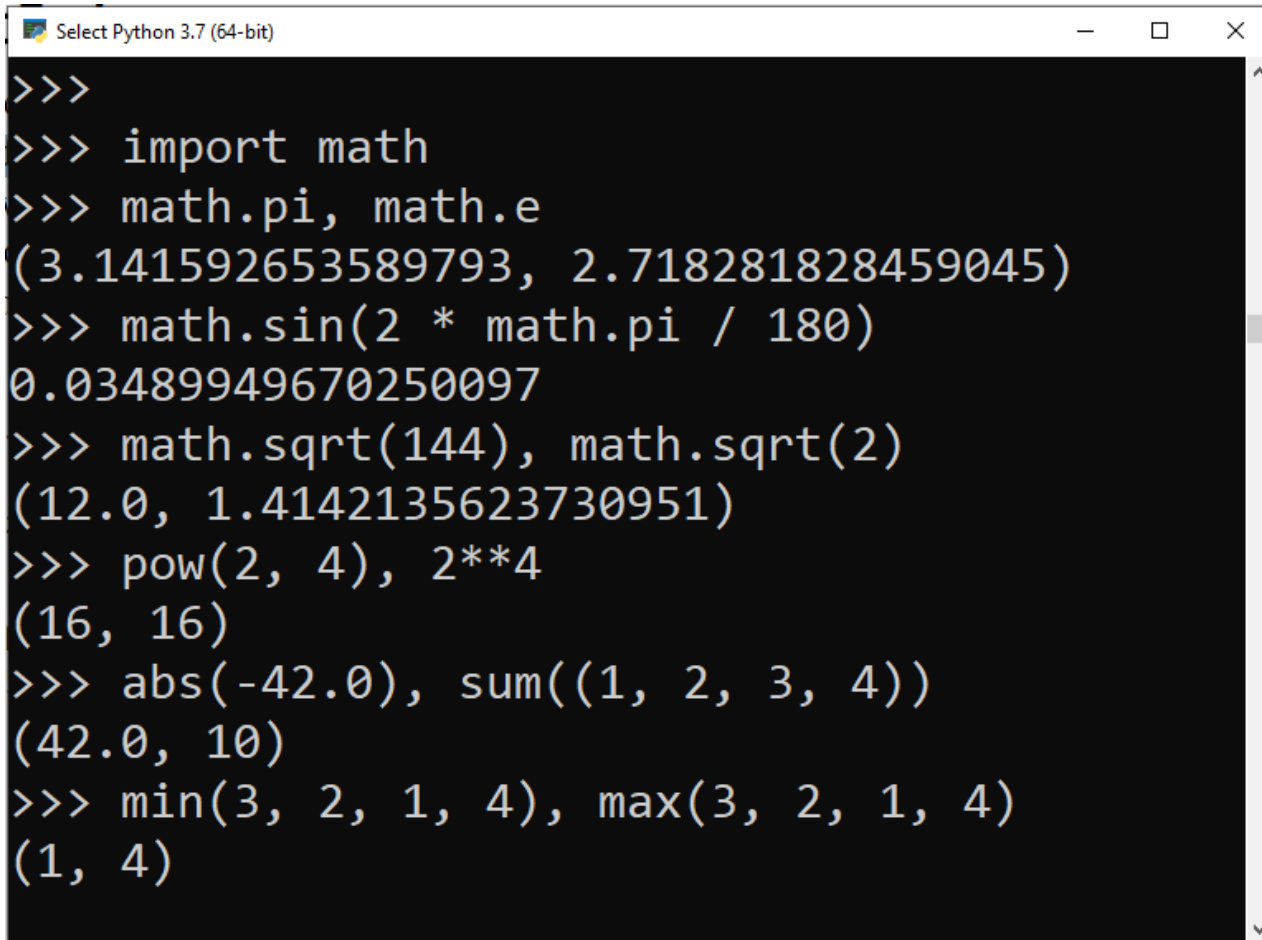
- Python supports most of the numeric expressions available in the C language.
- This includes operators that treat integers as strings of binary bits.



```
Python 3.7 (64-bit)
>>>
>>> x = 1
>>> x << 2, bin(x << 2)
(4, '0b100')
>>> x | 2, bin(x | 2)
(3, '0b11')
>>> x & 1, bin(x & 1)
(1, '0b1')
>>> x >> 2, bin(x >> 2)
(0, '0b0')
>>>
```

3. Other Numeric Types

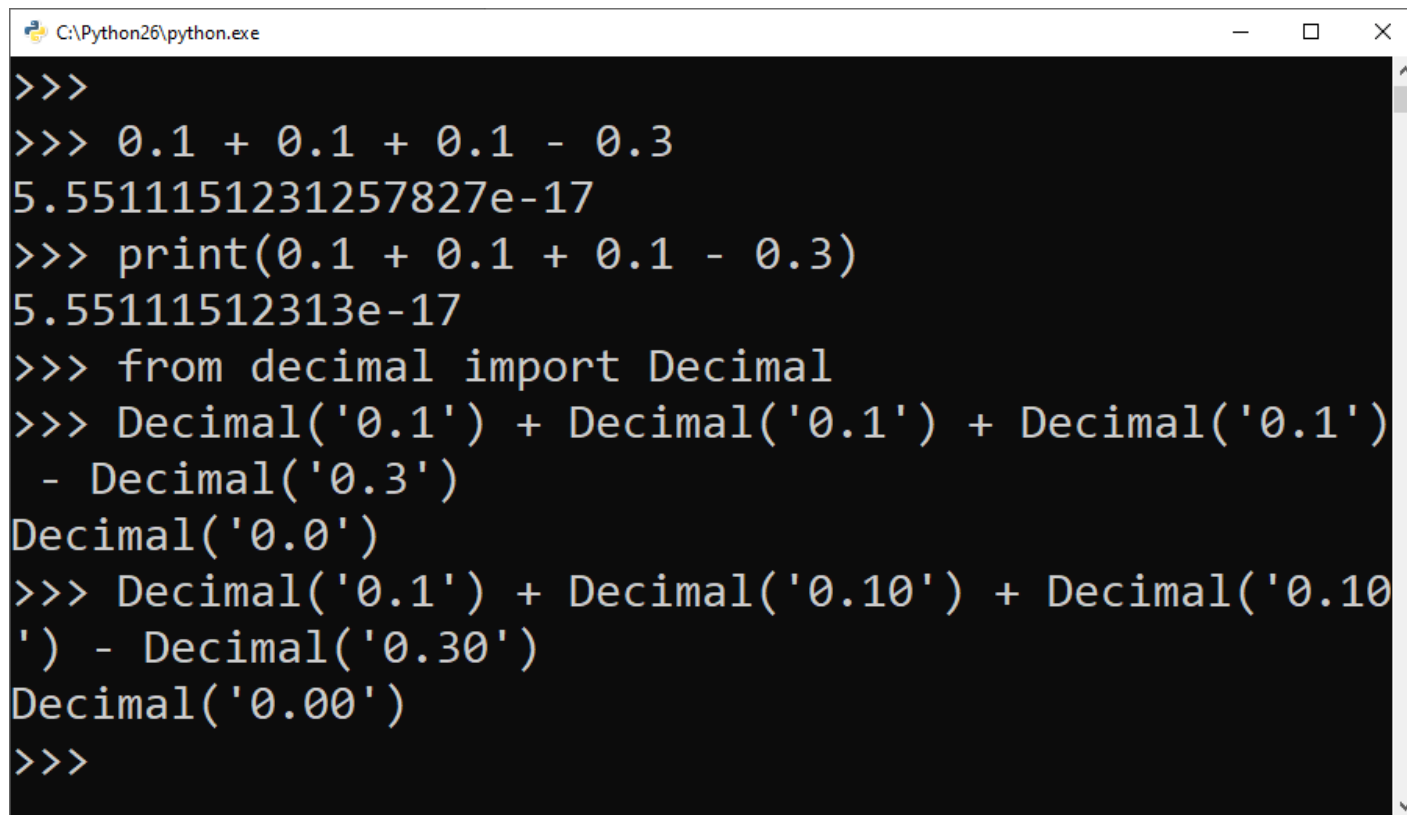
- Python also provides both built-in functions and standard library modules for numeric processing.

A screenshot of a Python 3.7 (64-bit) shell window. The window has a title bar that says "Select Python 3.7 (64-bit)" and standard window controls (minimize, maximize, close). The background is black, and the text is white. The code entered in the shell is as follows:

```
>>>  
>>> import math  
>>> math.pi, math.e  
(3.141592653589793, 2.718281828459045)  
>>> math.sin(2 * math.pi / 180)  
0.03489949670250097  
>>> math.sqrt(144), math.sqrt(2)  
(12.0, 1.4142135623730951)  
>>> pow(2, 4), 2**4  
(16, 16)  
>>> abs(-42.0), sum((1, 2, 3, 4))  
(42.0, 10)  
>>> min(3, 2, 1, 4), max(3, 2, 1, 4)  
(1, 4)
```

3.1 Decimal Type

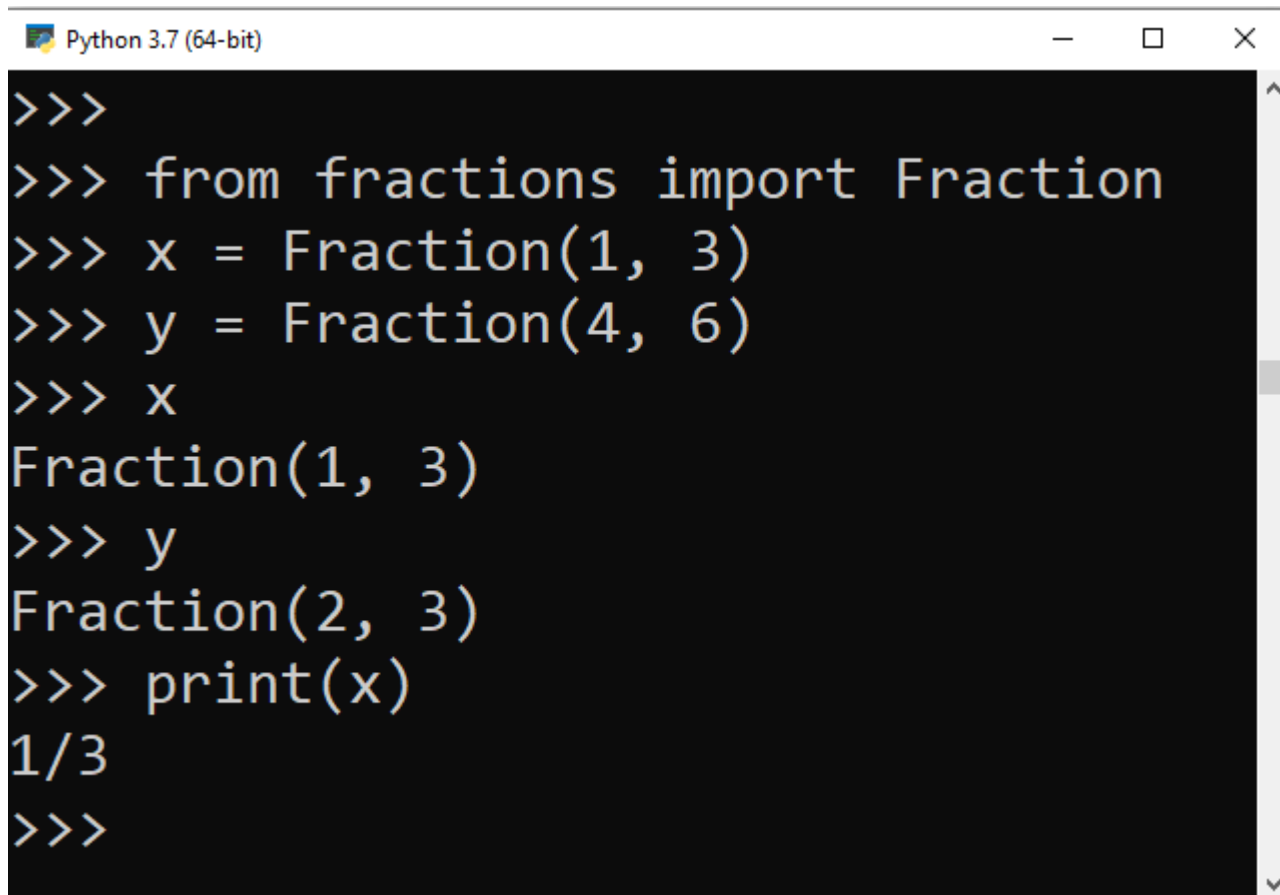
- Decimals are like **floating-point** numbers, but they have a **fixed number** of decimal points. Hence, decimals are **fixed-precision floating-point values**.



```
C:\Python26\python.exe
>>>
>>> 0.1 + 0.1 + 0.1 - 0.3
5.5511151231257827e-17
>>> print(0.1 + 0.1 + 0.1 - 0.3)
5.55111512313e-17
>>> from decimal import Decimal
>>> Decimal('0.1') + Decimal('0.1') + Decimal('0.1')
- Decimal('0.3')
Decimal('0.0')
>>> Decimal('0.1') + Decimal('0.10') + Decimal('0.10')
- Decimal('0.30')
Decimal('0.00')
>>>
```

3.2 Fraction Type

- It implements a **rational number** object. It essentially keeps both a numerator and a denominator explicitly.

A screenshot of a Python 3.7 (64-bit) shell window. The window has a title bar with the text "Python 3.7 (64-bit)" and standard window controls (minimize, maximize, close). The main area is a black terminal with yellow text. The code entered is: >>> from fractions import Fraction, >>> x = Fraction(1, 3), >>> y = Fraction(4, 6), >>> x, >>> y, and >>> print(x). The output shows Fraction(1, 3) for x and Fraction(2, 3) for y. The print statement outputs 1/3. The prompt >>> is shown at the end of the last line.

```
Python 3.7 (64-bit)
>>>
>>> from fractions import Fraction
>>> x = Fraction(1, 3)
>>> y = Fraction(4, 6)
>>> x
Fraction(1, 3)
>>> y
Fraction(2, 3)
>>> print(x)
1/3
>>>
```

3.2 Fraction Type (con't)

- Fractions can be used in **mathematical** expressions
- created from **floating-point number strings**

```
Python 3.7 (64-bit)
>>>
>>> x + y
Fraction(1, 1)
>>> x - y
Fraction(-1, 3)
>>> x * y
Fraction(2, 9)
>>>
```

```
Python 3.7 (64-bit)
>>>
>>> Fraction('0.25')
Fraction(1, 4)
>>> Fraction('1.25')
Fraction(5, 4)
>>> Fraction('0.25') + Fraction('1.25')
Fraction(3, 2)
>>>
```


3.3 Sets

- **unordered collection of unique and immutable objects** that supports operations corresponding to **mathematical** set theory.

```
Python 3.7 (64-bit)
>>>
>>> x = set('abcde')
>>> y = set('bdxyz')
>>> z = set('hello_hello')
>>> x
{'d', 'a', 'e', 'b', 'c'}
>>> y
{'d', 'z', 'x', 'y', 'b'}
>>> z
{'h', 'o', 'e', '_', 'l'}
>>>
```

3.3 Sets (con't)

Common mathematical set operations with expression operators

Membership

```
Python 3.7 (64-bit)
>>>
>>> 'e' in x
True
>>> x - y
{'c', 'a', 'e'}
>>> x | y
{'d', 'a', 'e', 'z', 'x', 'y', 'b', 'c'}
>>> x & y
{'b', 'd'}
>>> x ^ y
{'z', 'c', 'x', 'y', 'a', 'e'}
>>> x > y, x < y
(False, False)
>>>
```

3.3 Sets (con't)

Common mathematical set operations with expression operators

Difference

```
Python 3.7 (64-bit)
>>>
>>> 'e' in x
True
>>> x - y
{'c', 'a', 'e'}
>>> x | y
{'d', 'a', 'e', 'z', 'x', 'y', 'b', 'c'}
>>> x & y
{'b', 'd'}
>>> x ^ y
{'z', 'c', 'x', 'y', 'a', 'e'}
>>> x > y, x < y
(False, False)
>>>
```

3.3 Sets (con't)

Common mathematical set operations with expression operators

Union

```
Python 3.7 (64-bit)
>>>
>>> 'e' in x
True
>>> x - y
{'c', 'a', 'e'}
>>> x | y
{'d', 'a', 'e', 'z', 'x', 'y', 'b', 'c'}
>>> x & y
{'b', 'd'}
>>> x ^ y
{'z', 'c', 'x', 'y', 'a', 'e'}
>>> x > y, x < y
(False, False)
>>>
```

3.3 Sets (con't)

Common mathematical set operations with expression operators

Intersection

```
Python 3.7 (64-bit)
>>>
>>> 'e' in x
True
>>> x - y
{'c', 'a', 'e'}
>>> x | y
{'d', 'a', 'e', 'z', 'x', 'y', 'b', 'c'}
>>> x & y
{'b', 'd'}
>>> x ^ y
{'z', 'c', 'x', 'y', 'a', 'e'}
>>> x > y, x < y
(False, False)
>>>
```

3.3 Sets (con't)

Common mathematical set operations with expression operators

**Symmetric
difference (XOR)**

```
Python 3.7 (64-bit)
>>>
>>> 'e' in x
True
>>> x - y
{'c', 'a', 'e'}
>>> x | y
{'d', 'a', 'e', 'z', 'x', 'y', 'b', 'c'}
>>> x & y
{'b', 'd'}
>>> x ^ y
{'z', 'c', 'x', 'y', 'a', 'e'}
>>> x > y, x < y
(False, False)
>>>
```

3.3 Sets (con't)

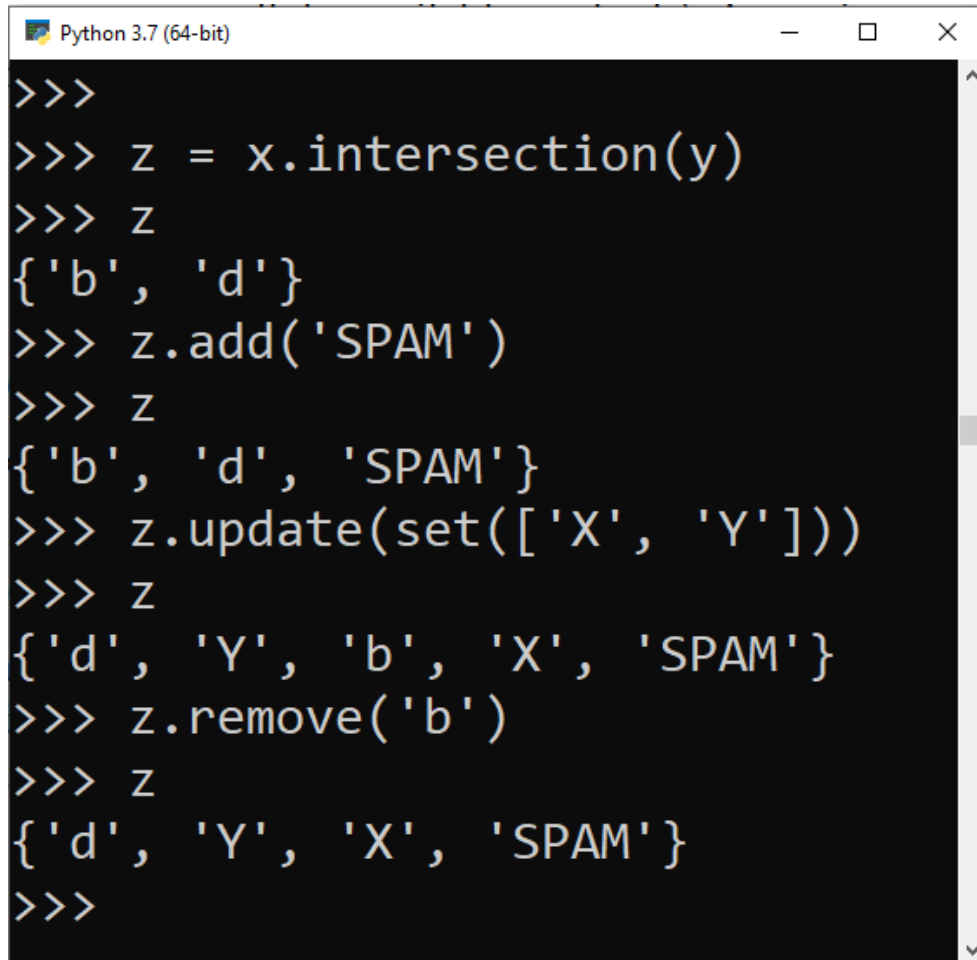
Common mathematical set operations with expression operators

Superset, subset

```
Python 3.7 (64-bit)
>>>
>>> 'e' in x
True
>>> x - y
{'c', 'a', 'e'}
>>> x | y
{'d', 'a', 'e', 'z', 'x', 'y', 'b', 'c'}
>>> x & y
{'b', 'd'}
>>> x ^ y
{'z', 'c', 'x', 'y', 'a', 'e'}
>>> x > y, x < y
(False, False)
>>>
```

3.3 Sets (con't)

- set object provides methods that correspond to these operations and more: add and remove

A screenshot of a Python 3.7 (64-bit) terminal window. The window has a title bar with the text "Python 3.7 (64-bit)" and standard window controls (minimize, maximize, close). The terminal background is black with white text. The code being executed is as follows:

```
>>>  
>>> z = x.intersection(y)  
>>> z  
{'b', 'd'}  
>>> z.add('SPAM')  
>>> z  
{'b', 'd', 'SPAM'}  
>>> z.update(set(['X', 'Y']))  
>>> z  
{'d', 'Y', 'b', 'X', 'SPAM'}  
>>> z.remove('b')  
>>> z  
{'d', 'Y', 'X', 'SPAM'}  
>>>
```


3.4 Booleans

- bool, is **numeric in nature** because its two values, **True** and **False**, are just customized versions of the integers **1** and **0**.

```
Python 3.7 (64-bit)
>>>
>>> type(True)
<class 'bool'>
>>> isinstance(True, int)
True
>>> True == 1
True
>>> True is 1
False
>>> True or False
True
>>> True + 4
5
```

Type True is bool

3.4 Booleans

- bool, is **numeric in nature** because its two values, **True** and **False**, are just customized versions of the integers **1** and **0**.

```
Python 3.7 (64-bit)
>>>
>>> type(True)
<class 'bool'>
>>> isinstance(True, int)
True
>>> True == 1
True
>>> True is 1
False
>>> True or False
True
>>> True + 4
5
```

True instance of
integer

3.4 Booleans

- bool, is **numeric in nature** because its two values, **True** and **False**, are just customized versions of the integers **1** and **0**.

```
Python 3.7 (64-bit)
>>>
>>> type(True)
<class 'bool'>
>>> isinstance(True, int)
True
>>> True == 1
True
>>> True is 1
False
>>> True or False
True
>>> True + 4
5
```

Same value

3.4 Booleans

- bool, is **numeric in nature** because its two values, **True** and **False**, are just customized versions of the integers **1** and **0**.

```
Python 3.7 (64-bit)
>>>
>>> type(True)
<class 'bool'>
>>> isinstance(True, int)
True
>>> True == 1
True
>>> True is 1
False
>>> True or False
True
>>> True + 4
5
```

But different object

Number Type Conversion:

- Type **int(x)** to convert x to a plain integer.
- Type **float(x)** to convert x to a floating-point number.
- Type **complex(x)** to convert x to a complex number with real part x and imaginary part zero.
- Type **complex(x, y)** to convert x and y to a complex number with real part x and imaginary part y. x and y are numeric expressions

Mathematical Functions:

Function	Returns (description)
<u>abs(x)</u>	The absolute value of x: the (positive) distance between x and zero.
<u>ceil(x)</u>	The ceiling of x: the smallest integer not less than x
<u>cmp(x, y)</u>	-1 if $x < y$, 0 if $x == y$, or 1 if $x > y$
<u>exp(x)</u>	The exponential of x: e^x
<u>fabs(x)</u>	The absolute value of x.
<u>floor(x)</u>	The floor of x: the largest integer not greater than x
<u>log(x)</u>	The natural logarithm of x, for $x > 0$
<u>log10(x)</u>	The base-10 logarithm of x for $x > 0$.

Mathematical Functions:

Function	Returns (description)
<u>max(x1, x2,...)</u>	The largest of its arguments: the value closest to positive infinity
<u>min(x1, x2,...)</u>	The smallest of its arguments: the value closest to negative infinity
<u>modf(x)</u>	The fractional and integer parts of x in a two-item tuple. Both parts have the same sign as x. The integer part is returned as a float.
<u>pow(x, y)</u>	The value of $x^{**}y$.
<u>round(x [,n])</u>	x rounded to n digits from the decimal point. Python rounds away from zero as a tie-breaker: round(0.5) is 1.0 and round(-0.5) is -1.0.
<u>sqrt(x)</u>	The square root of x for $x > 0$

Random Number Functions:

Function	Returns (description)
<u>choice(seq)</u>	A random item from a list, tuple, or string.
<u>randrange ([start,] stop [,step])</u>	A randomly selected element from range(start, stop, step)
<u>random()</u>	A random float r, such that 0 is less than or equal to r and r is less than 1
<u>seed([x])</u>	Sets the integer starting value used in generating random numbers. Call this function before calling any other random module function. Returns None.
<u>shuffle(lst)</u>	Randomizes the items of a list in place. Returns None.
<u>uniform(x, y)</u>	A random float r, such that x is less than or equal to r and r is less than y

Trigonometric Functions:

Function	Description
<u>acos(x)</u>	Return the arc cosine of x, in radians.
<u>asin(x)</u>	Return the arc sine of x, in radians.
<u>atan(x)</u>	Return the arc tangent of x, in radians.
<u>atan2(y, x)</u>	Return $\text{atan}(y / x)$, in radians.
<u>cos(x)</u>	Return the cosine of x radians.
<u>hypot(x, y)</u>	Return the Euclidean norm, $\text{sqrt}(x^2 + y^2)$.
<u>sin(x)</u>	Return the sine of x radians.
<u>tan(x)</u>	Return the tangent of x radians.
<u>degrees(x)</u>	Converts angle x from radians to degrees.
<u>radians(x)</u>	Converts angle x from degrees to radians.

Mathematical Constants:

Constant	Description
pi	The mathematical constant pi.
e	The mathematical constant e.
tau	The math.tau constant returns the value tau: 6.283185307179586. It is defined as the ratio of the circumference to the radius of a circle.
inf	The math.inf constant returns of positive infinity.
nan	The math.nan constant returns a floating-point nan (Not a Number) value.

Test Your Knowledge: Quiz

1. What is the value of the expression $2 * (3 + 4)$ in Python?
2. What is the value of the expression $2 * 3 + 4$ in Python?
3. What is the value of the expression $2 + 3 * 4$ in Python?
4. What tools can you use to find a number's square root, as well as its square?
5. What is the type of the result of the expression $1 + 2.0 + 3$?
6. How can you truncate and round a floating-point number?
7. How can you convert an integer to a floating-point number?
8. How would you display an integer in octal, hexadecimal, or binary notation?
9. How might you convert an octal, hexadecimal, or binary string to a plain integer?