

Variables, Operators, Data Types

講者：Isaac

Outline

- ▶ Variables
- ▶ Operators
- ▶ Data Types



Variables



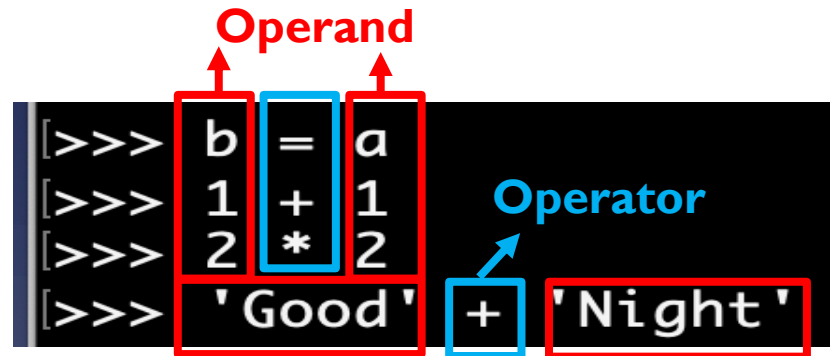
Variables

- ▶ A variable is a reserved memory location to store values.
 - ▶ Give data to the computer for processing
 - ▶ Do not need to be declared with any particular type and can even change type after they have been set.
- ▶ Use Equal sign “=” to assign values to a variable
 - ▶ The declaration happens automatically after assign values to variables.
 - ▶ For Example: Value 9 is assigned to Variable a

The diagram shows a code editor with a black background and white text. The code is `[>>> a = 9`. A red arrow points from the text "Variable Name" to the variable `a`. Another red arrow points from the text "Value" to the number `9`. A blue curved arrow points from the number `9` to the variable `a`, with the word "Assign" written in blue below it.

Assignment

- ▶ Assignment statement contains operand and operator.

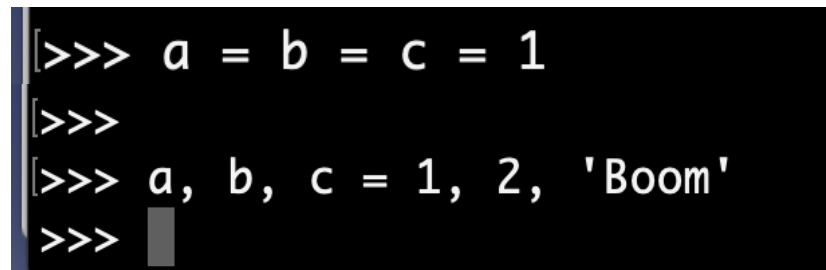


The diagram shows a Python interpreter session with four lines of code. Red boxes highlight the operands and the equals sign in the first line, and the entire first operand in the fourth line. A red arrow points to the equals sign, and another red arrow points to the second operand in the first line. A blue box highlights the plus sign in the second line, and a blue arrow points to it from the label 'Operator'. A blue box also highlights the plus sign in the fourth line. The code is as follows:

```
[>>> b = a  
[>>> 1 + 1  
[>>> 2 * 2  
[>>> 'Good' + 'Night']
```

- ▶ Multiple Assignment

- ▶ Python allows you to assign a single value to several variables simultaneously

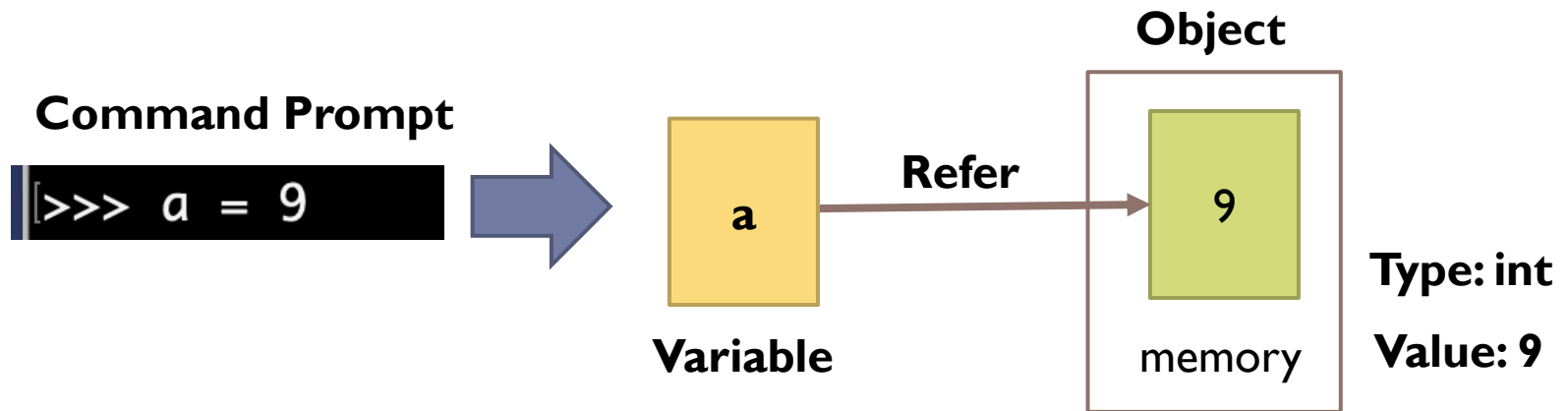


The diagram shows a Python interpreter session with four lines of code. The first line assigns the value 1 to variables a, b, and c. The second line is a blank prompt. The third line assigns the values 1, 2, and 'Boom' to variables a, b, and c respectively. The fourth line is a blank prompt. The code is as follows:

```
[>>> a = b = c = 1  
[>>>  
[>>> a, b, c = 1, 2, 'Boom'  
>>>
```

Object

- ▶ In Python, everything(booleans, integers, floats, strings, even large data structures, functions, and programs) is implemented as an object.
- ▶ An object is like a box that contains a piece of data. The object has a type, such as boolean or integer, that determines what can be done with the data.



Object

- ▶ Object contains identity, type, value
 - ▶ `identity(id)`: returns the identity of an object as an unchangeable integer. This integer usually corresponds to the object's location in memory.
 - ▶ `type`: returns the type of an object, also is unchangeable.
 - ▶ `value`: returns the value of an object, can be either mutable or immutable.

identity

```
[>>> a = 'Hello'
>>> b = 'Hello'
>>> id(a)
4552193072
>>> id(b)
4552193072
```

type

```
[>>> type(a)
<class 'str'>
>>> c = 10
>>> type(c)
<class 'int'>
```

value

```
[>>> d = 22.11
>>> print(d)
22.11
```

Variables

▶ Dynamic Typing

- ▶ Python is a **dynamically typed** language.
- ▶ Python interpreter does type checking only as code runs, and the type of a variable is allowed to change over its lifetime.

```
>>> variable_A = 'Hello'
>>> type(variable_A)
<class 'str'>
>>> variable_A = 9527.87
>>> type(variable_A)
<class 'float'>
>>>
```

▶ Static Typing

- ▶ C/Java are statically typed languages.
- ▶ Program runs after compiler do the static type checks and the type of a variable is not allowed to change over its lifetime.

Operators



Operators

- ▶ Python operator falls into 7 categories:
 - ▶ Arithmetic Operator
 - ▶ Comparison Operator
 - ▶ Assignment Operator
 - ▶ Logical Operator
 - ▶ Membership Operator
 - ▶ Identity Operator
 - ▶ Bitwise Operator

Arithmetic Operators

Operator	Name	Example
+	Addition	$x + y$
-	Subtraction	$x - y$
*	Multiplication	$x * y$
/	Division	x / y
%	Modulus	$x \% y$
**	Exponentiation	$x ** y$
//	Floor division	$x // y$

1	25 + 5
30	
1	5 - 10
-5	
1	3 * 3
9	
1	10 / 3
3.3333333333333335	
1	10 % 3
1	
1	3 ** 3
27	
1	10 // 3
3	

Comparison Operators

Operator	Example	Description
==	x == y	If the values of two operands are equal, then the condition becomes true

```
1 x = 10
2 y = 2
```

```
1 x == y
```

False

```
1 # please focus on the comparsion operator
2 # ignore the if-else statement syntax.
3 if (x == y):
4     print('x is equal to y.')
5 else:
6     print('x is not equal to y.')
```

x is not equal to y.

Comparison Operators

Operator	Example	Description
<code>!=</code>	<code>x != y</code>	If values of two operands are not equal, then condition becomes true

```
1 x = 10
2 y = 2
```

```
1 x != y
```

True

```
1 # please focus on the comparsion operator
2 # ignore the if-else statement syntax.
3 if (x != y):
4     print('x is not equal to y.')
5 else:
6     print('x is equal to y.')
```

x is not equal to y.

Comparison Operators

Operator	Example	Description
>	$x > y$	If the value of left operand is greater than the value of right operand, then condition becomes true.

```
1 x = 10
2 y = 2
```

```
1 x > y
```

True

```
1 # please focus on the comparsion operator
2 # ignore the if-else statement syntax.
3 if (x > y):
4     print('x is greater than y.')
5 else:
6     print('x is not greater than y.')
```

x is greater than y.

Comparison Operators

Operator	Example	Description
<	$x < y$	If the value of left operand is less than the value of right operand, then condition becomes true.

```
1 x = 10
2 y = 2
```

```
1 x < y
```

False

```
1 # please focus on the comparsion operator
2 # ignore the if-else statement syntax.
3 if (x < y):
4     print('x is less than y.')
5 else:
6     print('x is not less than y.')
```

x is not less than y.

Comparison Operators

Operator	Example	Description
>=	x >= y	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.

```
1 x = 10
2 y = 2
```

```
1 x >= y
```

True

```
1 # please focus on the comparsion operator
2 # ignore the if-else statement syntax.
3 if (x >= y):
4     print('x is greater than or equal to y.')
5 else:
6     print('x is not greater than or equal to y.')
```

x is greater than or equal to y.

Comparison Operators

Operator	Example	Description
<=	x <= y	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.

```
1 x = 10
2 y = 2
```

```
1 x <= y
```

False

```
1 # please focus on the comparsion operator
2 # ignore the if-else statement syntax.
3 if (x <= y):
4     print('x is less than or equal to y.')
5 else:
6     print('x is not less than or equal to y.')
```

x is not less than or equal to y.

Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
=	x = 3	x = x 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Logical Operators

Operator	Description	Example
and	Returns True if both statements are true	x and y
or	Returns True if one of the statements is true	x or y
not	Reverse the result, returns False if the result is true	not(x and y)

```
1 x = True
2 y = False
```

```
1 x and y
```

False

```
1 x or y
```

True

```
1 not (x and y )
```

True

Membership Operators

Operator	Description	Example	Result
in	Returns True if a sequence with the specified value is present in the object	x in y	<pre>[>>> x = 3 [>>> y = [1,2,3,4,5] [>>> x in y True [>>> x not in y False</pre>
not in	Returns True if a sequence with the specified value is not present in the object	x not in y	

Identity Operators

- ▶ Check values are located on the same part of the memory.
- ▶ Variables with same value does not mean identical.

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

```
1 x1 = 2
2 y1 = 2
3
4 print(x1 is y1)
```

True

```
1 x2 = 'Hello'
2 y2 = 'Python'
3
4 print(x2 is y2)
```

False

```
1 x3 = [2, 'hello']
2 y3 = [2, 'hello']
3
4 print(x3 is y3)
```

False

Bitwise Operators

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits

Example:

$x = 10 = 0b1010$ (Binary)

$y = 4 = 0b0100$ (Binary)

$x \& y = 0b1010 \& 0b0100 = 0b0000 = 0$ (Decimal)

$x | y = 0b1010 | 0b0100 = 0b1110 = 14$ (Decimal)

$x \wedge y = 0b1010 \wedge 0b0100 = 0b1110 = 14$ (Decimal)

$\sim x = \sim 0b1010 = -0b1011 = -11$ (Decimal)

Bitwise Operators

Operator	Name	Description
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Example:

$x = 10 = 0b0000\ 1010$ (Binary)

$x \ll 2 = 0b0010\ 1000 = 40$ (Decimal)

$x \gg 2 = 0b0000\ 0010 = 2$ (Decimal)

Operator Precedence

Highest	Operator	Example
	<code>**</code>	Exponentiation
	<code>~x</code>	Bitwise not
	<code>+x, -x</code>	Positive, negative
	<code>*, /, %</code>	Multiplication, division, remainder
	<code>+, -</code>	Addition, subtraction
	<code><<, >></code>	Bitwise shifts
	<code>&</code>	Bitwise AND
	<code>^</code>	Bitwise XOR
	<code> </code>	Bitwise OR
	<code>in, not in, is, is not, <, <=, >, >=, <>, !=, ==</code>	Comparisons, membership, identity
	<code>not x</code>	Boolean NOT
	<code>and</code>	Boolean AND
Lowest	<code>or</code>	Boolean OR

Data Types



Python3 Standard Data Types

▶ Numeric Type

- ▶ int: 2, 5, 987
- ▶ float: 2.34, 5.67e-3(5.67×10^{-3})
- ▶ complex: 8+9j

▶ Boolean Type

- ▶ bool: True(1), False(0)

▶ Sequence Type

- ▶ String, Tuple, List

▶ Hashable Type

- ▶ Set, Dictionary

Mutable vs. Immutable

	Mutable	Immutable
value	Pass by reference	Pass by value(object address)
Data Types	list	numeric
	set	string
	dictionary	tuple

Numeric Data Types

▶ Integer:

- ▶ Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.
- ▶ During calculation, the int will convert into float.

▶ Float:

- ▶ Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

▶ Complex:

- ▶ Complex numbers are written with a "j" as the imaginary part.
- ▶ .real returns real part, .imag returns imaginary part.

Numeric Data Types

int	float	complex
5	0.0	5+3j
22	22.5	5.j
-3	-3.89	-3.89j
0b1010	-3.	.3j
-0b1011	18.3e+9	18.3e+9j
0x220	-18.3e+9	18.3e-9j
-0x220		

Boolean Data Types

- ▶ Booleans represent one of two values: **True** or **False**.
- ▶ In programming you often need to know if an expression is True or False.
- ▶ You can evaluate any expression in Python, and get one of two answers, True or False.
- ▶ When you compare two values, the expression is evaluated and Python returns the Boolean answer:

```
[>>> print(5>3)
True
[>>> print(5<3)
False
[>>> type(2<3)
<class 'bool'>
```

Sequence Data Types

- ▶ String
- ▶ List
- ▶ Tuple

String

- ▶ Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single (") or double quotes("").
- ▶ Short String:

```
>>> s = 'Hello'
```

```
>>> s2 = "!!!"
```

- ▶ Long String:

```
>>> '''Once upon a time,  
... i meet a beautiful woman,  
... named Julia.'''  
'Once upon a time, \ni meet a beautiful woman, \nnamed Julia.'  
>>>
```


String

▶ String Indexing with [] operator:

Example: s = 'Hello'

String	H	e	l	l	o
Index from head	0	1	2	3	4
Index from tail	-5	-4	-3	-2	-1

```
>>> s = 'Hello'
>>> s[0]
'H'
>>> s[1]
'e'
>>> s[2]
'l'
>>> s[-1]
'o'
```

▶ String Slicing:

▶ s[start : end : interval]

```
>>> s = 'python'
>>> s[2:]
'thon'
>>> s[:-2]
'pyth'
>>> s[1:-1:2]
'yh'
```

String

- ▶ String Concatenating with '+' operator:
- ▶ String Repeating with '*' operator:

```
>>> s = 'Hello'
>>> s2 = "!!!"
>>> s + s2
'Hello!!!'
>>> s + s2*2
'Hello!!!!!!'
>>> 
```

- ▶ String Escaping:
 - ▶ Use '\' to escape specific character.

```
>>> s1 = 'Good Morning'
>>> s2 = 'sir.'
>>> s3 = s1 + '\n' + s2
>>> print(s3)
Good Morning
sir.
```

Common String Operations

Operation	Description
<code>s[i]</code>	Index <code>i</code> of string <code>s</code>
<code>s[i:j]</code>	Slice from index <code>i</code> to <code>j</code>
<code>s[i:j:k]</code>	Slice from index <code>i</code> to <code>j</code> with interval <code>k</code>
<code>s + s2</code>	Connect string <code>s</code> with string <code>s2</code>
<code>s*n</code> or <code>n*s</code>	Multiply <code>n</code> times of string <code>s</code>
<code>len(s)</code>	Length of string <code>s</code>
<code>min(s)</code>	The minimum value of string <code>s</code>
<code>max(s)</code>	The maximum value of string <code>s</code>
<code>x not in s</code>	If object <code>x</code> is not in the string <code>s</code> , return <code>True</code> .
<code>x in s</code>	If object <code>x</code> is in the string <code>s</code> , return <code>True</code> .

Python3 Built-in String Methods

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits

Example

```
1 # str.capitalize()
2 txt = 'hello, python.'
3 txt.capitalize()
```

'Hello, python.'

```
1 # str.casefold()
2 txt = 'HELLO, PYTHON.'
3 txt.casefold()
```

'hello, python.'

```
1 # str.center(width, [fillchar])
2 txt = 'hello, python.'
3 txt.center(20, '-')
```

'---hello, python.---

```
1 # str.count(sub, start = 0, end = len(string))
2 txt = 'hello, python.'
3 txt.count('o', 0, 40)
```

2

Example

```
1 # str.encode
2 txt = '程式設計'
3 txt.encode('UTF-8')
```

b'\xe7\xa8\x8b\xe5\xbc\x8f\xe8\xa8\xad\xe8\xa8\x88'

```
1 # str.endswith(suffix[, start[, end]])
2 txt = 'Hello, Python!!'
3 txt.endswith('!', 0, 20)
```

True

```
1 # str.expandtabs(tabsize = 8)
2 txt = 'H\tello, Python!!'
3 txt.expandtabs(8)
```

'H ello, Python!!'

```
1 # str.find(str, start = 0 end = len(string))
2 txt = 'Hello, Python.'
3 txt.find('Python')
```

7

```
1 # str.index(str, start = 0 end = len(string))
2 txt = 'Hello, Python.'
3 txt.index('Python')
```

7

String Formatting

► Format

```
: 1 # str.format
   2 # default arguments
   3 print("Hello, {}".format('python'))
   4
   5 # positional arguments
   6 print("{} {}".format('hello', 'python'))
```

Hello, python.
hello, python.

► Format_map

```
: 1 # str.format_map
   2 txt = {'x': 'hello', 'y': 'python'}
   3 print('{x}, {y}'.format(**txt))
```

hello, python

Python3 Built-in String Methods

Method	Description
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Joins the elements of an iterable to the end of the string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value

Example

```
: 1 # str.join(sequence)
2 txt = '-'
3 txt.join('hello,python.')
```

```
: 'h-e-l-l-o-, -p-y-t-h-o-n-.'
```

```
: 1 # str.replace
2 txt = 'hello, python.'
3 txt.replace('hello', 'hi')
```

```
: 'hi, python.'
```

```
: 1 # str.split
2 txt = 'hello, python.'
3 txt.split(',')
```

```
: ['hello', ' python.']
```

```
: 1 # str.lstrip([chars])
2 txt = '~~~~hello, python.~~~~'
3 txt.lstrip('~')
```

```
: 'hello, python.~~~~'
```

```
: 1 # str.strip
2 txt = '~~~~hello, python.~~~~'
3 txt.strip('~')
```

```
: 'hello, python.'
```

Python3 Built-in String Methods

Method	Description
<code>rfind()</code>	Searches the string for a specified value and returns the last position of where it was found
<code>rindex()</code>	Searches the string for a specified value and returns the last position of where it was found
<code>rjust()</code>	Returns a right justified version of the string
<code>rpartition()</code>	Returns a tuple where the string is parted into three parts
<code>rsplit()</code>	Splits the string at the specified separator, and returns a list
<code>rstrip()</code>	Returns a right trim version of the string
<code>split()</code>	Splits the string at the specified separator, and returns a list
<code>splitlines()</code>	Splits the string at line breaks and returns a list
<code>startswith()</code>	Returns true if the string starts with the specified value
<code>strip()</code>	Returns a trimmed version of the string
<code>swapcase()</code>	Swaps cases, lower case becomes upper case and vice versa
<code>title()</code>	Converts the first character of each word to upper case
<code>translate()</code>	Returns a translated string
<code>upper()</code>	Converts a string into upper case
<code>zfill()</code>	Fills the string with a specified number of 0 values at the beginning



Example

```
1 # str.lower
2 txt = 'HELLO, PYTHON.'
3 txt.lower()
```

```
: 'hello, python.'
```

```
1 # str.upper
2 txt = 'hello, python.'
3 txt.upper()
```

```
: 'HELLO, PYTHON.'
```

```
1 # str.swapcase
2 txt = 'hello, python.'
3 txt.swapcase()
```

```
: 'HELLO, PYTHON.'
```

List

- ▶ A collection which is ordered and **changeable**.
- ▶ Allows duplicate members.
- ▶ Elements in List can be different data types.
- ▶ In Python lists are written with **square** brackets.

Example: L = ['Hello', 'Hi', 'Hey', 'Yo', 'Sup']

List	Hello	Hi	Hey	Yo	Sup
Index from head	0	1	2	3	4
Index from tail	-5	-4	-3	-2	-1

Python3 Built-in List Methods

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Example

► List Slicing

```
1 # List + slicing
2 weekdays = ['Mon', 'Tue', 'Wed', 'Thurs', 'Fri']
3 weekdays[2:4]
```

['Wed', 'Thurs']

```
1 # List + slicing
2 weekdays = ['Mon', 'Tue', 'Wed', 'Thurs', 'Fri']
3 weekdays[1:]
```

['Tue', 'Wed', 'Thurs', 'Fri']

```
1 # List + slicing
2 weekdays = ['Mon', 'Tue', 'Wed', 'Thurs', 'Fri']
3 weekdays[::2]
```

['Mon', 'Wed', 'Fri']

Example

► Append

```
1 # List + append
2 weekdays.append('SAT')
3 print weekdays
```

['Mon', 'Tue', 'Wed', 'Thurs', 'Fri', 'SAT']

```
1 # List + append
2 weekdays = ['Mon', 'Tue', 'Wed', 'Thurs', 'Fri']
3 weekend = ['Sat', 'Sun']
4
5 weekdays.append(weekend)
6 print weekdays
```

['Mon', 'Tue', 'Wed', 'Thurs', 'Fri', ['Sat', 'Sun']]

► extend

```
: 1 # List + extend
   2 weekdays = ['Mon', 'Tue', 'Wed', 'Thurs', 'Fri']
   3 weekend = ['Sat', 'Sun']
   4
   5 weekdays.extend(weekend)
   6 print weekdays
```

['Mon', 'Tue', 'Wed', 'Thurs', 'Fri', 'Sat', 'Sun']

Example

► Insert

```
1 # List + insert
2 weekend = ['Sat', 'Sun']
3 weekend.insert(1, 'AA')
4 weekend
```

['Sat', 'AA', 'Sun']

► pop

```
1 # List + pop
2 weekdays = ['Mon', 'Tue', 'Wed', 'Thurs', 'Fri']
3 weekdays.pop()
```

'Fri'

```
1 weekdays.pop()
```

'Thurs'

```
1 weekdays
```

['Mon', 'Tue', 'Wed']

Example

► remove

```
: 1 # List + remove  
2 weekdays = ['Mon', 'Tue', 'Wed', 'Thurs', 'Fri']  
3 weekdays.remove('Mon')  
4 weekdays
```

```
: ['Tue', 'Wed', 'Thurs', 'Fri']
```

Tuple

- ▶ A collection which is ordered and **unchangeable**.
- ▶ Allows duplicate members.
- ▶ Elements in Tuple can be different data types.
- ▶ In Python tuples are written with **round** brackets.

Example: T = ('Hello', 'Hi', 33, 44, '55')

Tuple	Hello	Hi	33	44	55
Index from head	0	1	2	3	4
Index from tail	-5	-4	-3	-2	-1

Python3 Built-in Tuple Methods

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

```
1 txt = ('p', 'y', 't', 'h', 'o', 'n')  
2 txt.count('y')
```

1

```
1 txt = ('p', 'y', 't', 'h', 'o', 'n')  
2 txt.index('t')
```

2

Hashable Data Types

- ▶ Set
- ▶ Dictionary

Set

- ▶ A collection which is unordered and unindexed.
- ▶ No duplicate members.
- ▶ In Python sets are written with **curly** brackets.
- ▶ Generally used in mathematic set theory.

```
>>>  
>>> grade_set = {98, 97, 100, 59, 61}  
>>>
```

Python3 Built-in Set Methods

Method	Description
<code>add()</code>	Adds an element to the set
<code>clear()</code>	Removes all the elements from the set
<code>copy()</code>	Returns a copy of the set
<code>difference()</code>	Returns a set containing the difference between two or more sets
<code>difference_update()</code>	Removes the items in this set that are also included in another, specified set
<code>discard()</code>	Remove the specified item
<code>intersection()</code>	Returns a set, that is the intersection of two other sets
<code>intersection_update()</code>	Removes the items in this set that are not present in other, specified set(s)
<code>isdisjoint()</code>	Returns whether two sets have a intersection or not
<code>issubset()</code>	Returns whether another set contains this set or not
<code>issuperset()</code>	Returns whether this set contains another set or not

Python3 Built-in Set Methods

Method	Description
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

Dictionary

- ▶ A collection which is unordered, changeable and indexed.
- ▶ No duplicate members.
- ▶ In Python dictionaries are written with curly brackets, and they have keys and values.
 - ▶ Key(Immutable): use key as index to access items in dictionary.

Key **Value**

```
[>>> Dict = {'Jason':19, 'Mary':100, 'Igor':67}
```


Common Dictionary Operations

Operation	Description
<code>d[key]</code>	Access value of index key in dictionary d
<code>d[key] = value</code>	Assign value to index key in dictionary d
<code>del d[key]</code>	Delete value of index key in dictionary d
<code>iter(d)</code>	Return the iterator build by key.
<code>len(d)</code>	Length of dictionary d
<code>key not in s</code>	If key is not in dictionary d, return True.
<code>key in s</code>	If key is in dictionary d, return True.

Python3 Built-in Dictionary Methods

Method	Description
<code>clear()</code>	Removes all the elements from the dictionary
<code>copy()</code>	Returns a copy of the dictionary
<code>fromkeys()</code>	Returns a dictionary with the specified keys and value
<code>get()</code>	Returns the value of the specified key
<code>items()</code>	Returns a list containing a tuple for each key value pair
<code>keys()</code>	Returns a list containing the dictionary's keys
<code>pop()</code>	Removes the element with the specified key
<code>popitem()</code>	Removes the last inserted key-value pair
<code>setdefault()</code>	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
<code>update()</code>	Updates the dictionary with the specified key-value pairs
<code>values()</code>	Returns a list of all the values in the dictionary

Example

► Keys

```
1 # dict.keys
2 fruit_price = {'watermelon':100, 'apple':80, 'avocada':90, 'strawberry':150}
3 fruit_price.keys()

dict_keys(['watermelon', 'apple', 'avocada', 'strawberry'])
```

► values

```
1 # dict.values
2 fruit_price = {'watermelon':100, 'apple':80, 'avocada':90, 'strawberry':150}
3 fruit_price.values()

dict_values([100, 80, 90, 150])
```

Example

► Items

```
: 1 # dict.items
2 fruit_price = {'watermelon':100, 'apple':80, 'avocada':90, 'strawberry':150}
3 fruit_price.items()

: dict_items([('watermelon', 100), ('apple', 80), ('avocada', 90), ('strawberry', 150)])
```

► clear

```
: 1 # dict.clear
2 fruit_price = {'watermelon':100, 'apple':80, 'avocada':90, 'strawberry':150}
3 fruit_price.clear()
4
5 fruit_price

: {}
```

Casting

- ▶ There may be times when you want to specify a type on to a variable. This can be done with casting.
- ▶ Casting in python is therefore done using constructor functions:
 - ▶ `int()`: constructs an integer number from an integer literal, a float literal (by rounding down to the previous whole number), or a string literal (providing the string represents a whole number).
 - ▶ `float()`: constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer).
 - ▶ `str()`: constructs a string from a wide variety of data types, including strings, integer literals and float literals.

Python3 Built-in Casting Methods

Method	Description
<code>complex(real, [imag])</code>	Construct a complex value with real and imagery part.
<code>list(s)</code>	Convert string s into a new list.
<code>tuple(s)</code>	Convert string s into a new tuple.
<code>set(s)</code>	Convert string s into a new set.
<code>hex(x)</code>	Convert integer x to a hex number.
<code>oct(x)</code>	Convert integer x to a oct number.

Example

```
1 x = 100  
2 hex(x)
```

'0x64'

```
1 x = 100  
2 oct(x)
```

'0o144'

```
1 txt = 'python'  
2 list(txt)
```

['p', 'y', 't', 'h', 'o', 'n']

```
1 txt = 'python'  
2 tuple(txt)
```

('p', 'y', 't', 'h', 'o', 'n')