

物聯網與微處理機系統設計

Internet of Things and Microprocessor System Design

Lecture 01 – Python Programming

Lecturer: 陳彥安 Chen, Yan-Ann
YZU CSE

Outline





















- Python Basics
- Data Types
- Control Flow
- String
- Files I/O
- Modules
- Function & Class

Outline

- Python Basics
- Data Types
- Control Flow
- String
- Files I/O
- Modules
- Function & Class

PL Popularity (1/2)

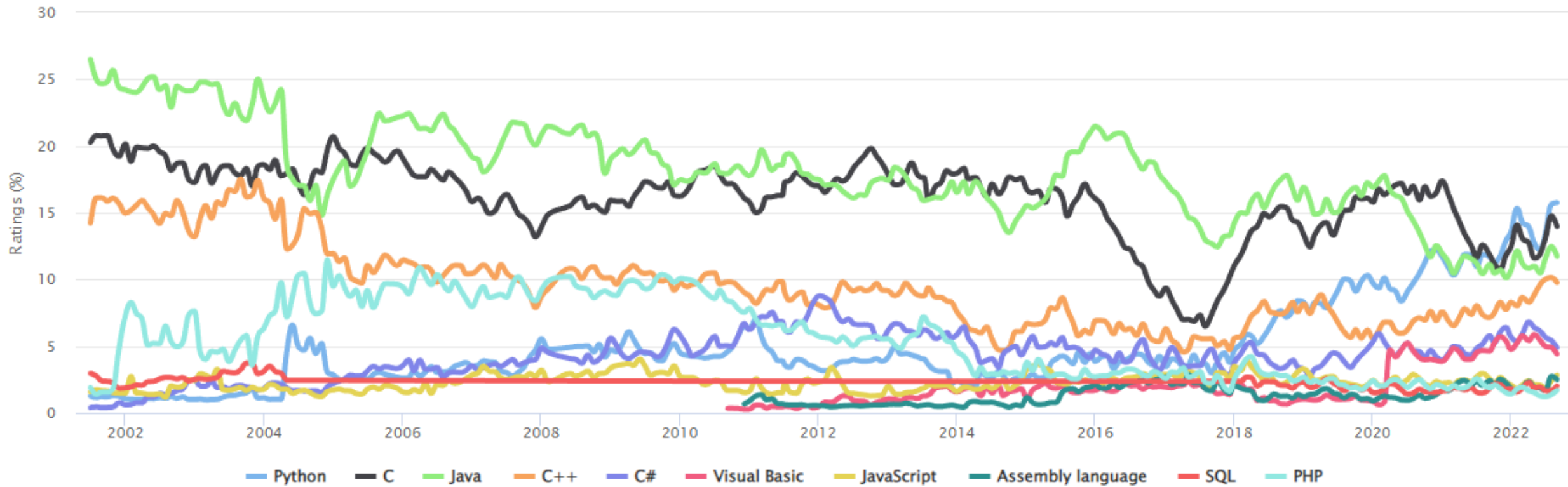
- TIOBE Index in September 2022
 - <https://www.tiobe.com/tiobe-index/>

Sep 2022	Sep 2021	Change	Programming Language		Ratings	Change
1	2	▲		Python	15.74%	+4.07%
2	1	▼		C	13.96%	+2.13%
3	3			Java	11.72%	+0.60%
4	4			C++	9.76%	+2.63%
5	5			C#	4.88%	-0.89%
6	6			Visual Basic	4.39%	-0.22%
7	7			JavaScript	2.82%	+0.27%
8	8			Assembly language	2.49%	+0.07%
9	10	▲		SQL	2.01%	+0.21%
10	9	▼		PHP	1.68%	-0.17%
11	24	▲▲		Objective-C	1.49%	+0.86%
12	14	▲		Go	1.16%	+0.03%
13	20	▲▲		Delphi/Object Pascal	1.09%	+0.32%
14	16	▲		MATLAB	1.06%	+0.04%
15	17	▲		Fortran	1.03%	+0.02%
16	15	▼		Swift	0.98%	-0.09%
17	11	▼▼		Classic Visual Basic	0.98%	-0.55%
18	18			R	0.95%	-0.02%
19	19			Perl	0.72%	-0.06%
20	13	▼▼		Ruby	0.66%	-0.62%

PL Popularity (2/2)

TIOBE Programming Community Index

Source: www.tiobe.com

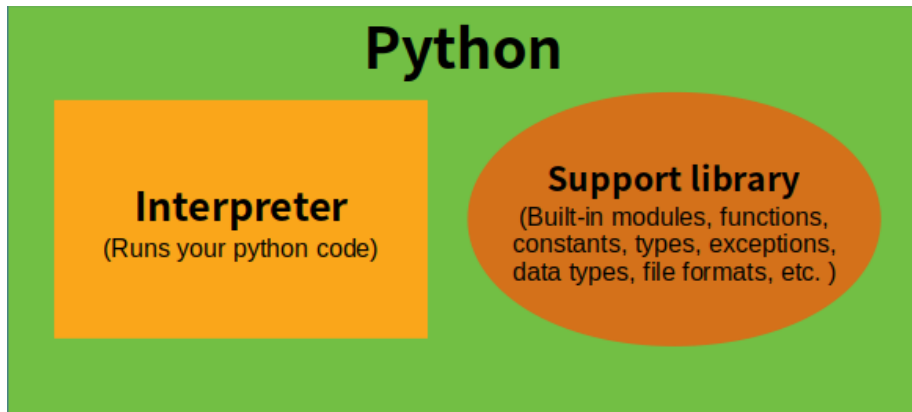


Introduction

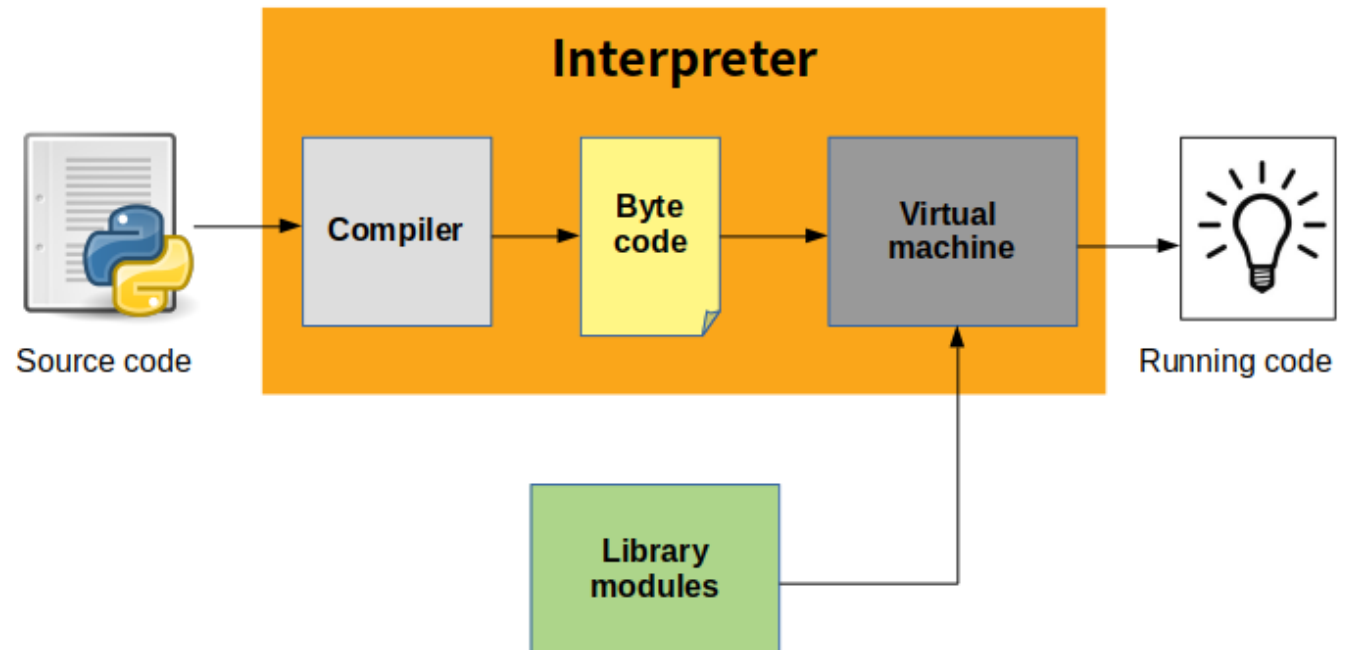
- Python is a clear and powerful object-oriented programming language
 - Uses an elegant syntax, making the programs you write easier to read.
 - Is an easy-to-use language that makes it simple to get your program working.
 - Comes with [a large standard library](#) that supports many common programming tasks.
 - Python's [interactive mode](#) makes it easy to test short snippets of code.
 - Is easily extended by adding new modules implemented in a compiled language such as C or C++.
 - Can also be embedded into an application to provide a programmable interface.
 - [Runs anywhere](#), including Mac OS X, Windows, Linux, and Unix.
 - Is free software in two senses. It doesn't cost anything to download or use Python, or to include it in your application.

Interpreter

Installed on your machine:



Run on your machine:



Ref: <https://www.quora.com/How-does-a-Python-program-get-executed-Is-there-any-concept-of-compile-time-and-run-time-in-Python>

Execution Types (1/3)

- Python Interactive Shell

```
Python 3.8.3 (default, Jul 2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

- You can type things directly into a running Python session.

```
>>> A = 1
>>> B = 2
>>> C = A + B
>>> C
3
>>> c
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'c' is not defined
```

Python is case-sensitive.

```
>>> name = "yzu"
>>> name
'yzu'
>>> print("hello", name)
hello yzu
>>> print("hello %s" % name)
hello yzu
```


Execution Types (2/3)

- Run with a saved file.

```
hello.py  ×
D: > hello.py > ...
1  A = 1
2  B = 2
3  C = A + B
4  print(C)
5  name = "yzu"
6  print("hello", name)
```

- Run module
 - You have to change the directory to the one where the hello.py is in.

```
(base) D:\>python
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import hello
3
hello yzu
```

Execution Types (3/3)

- Run with a saved file.

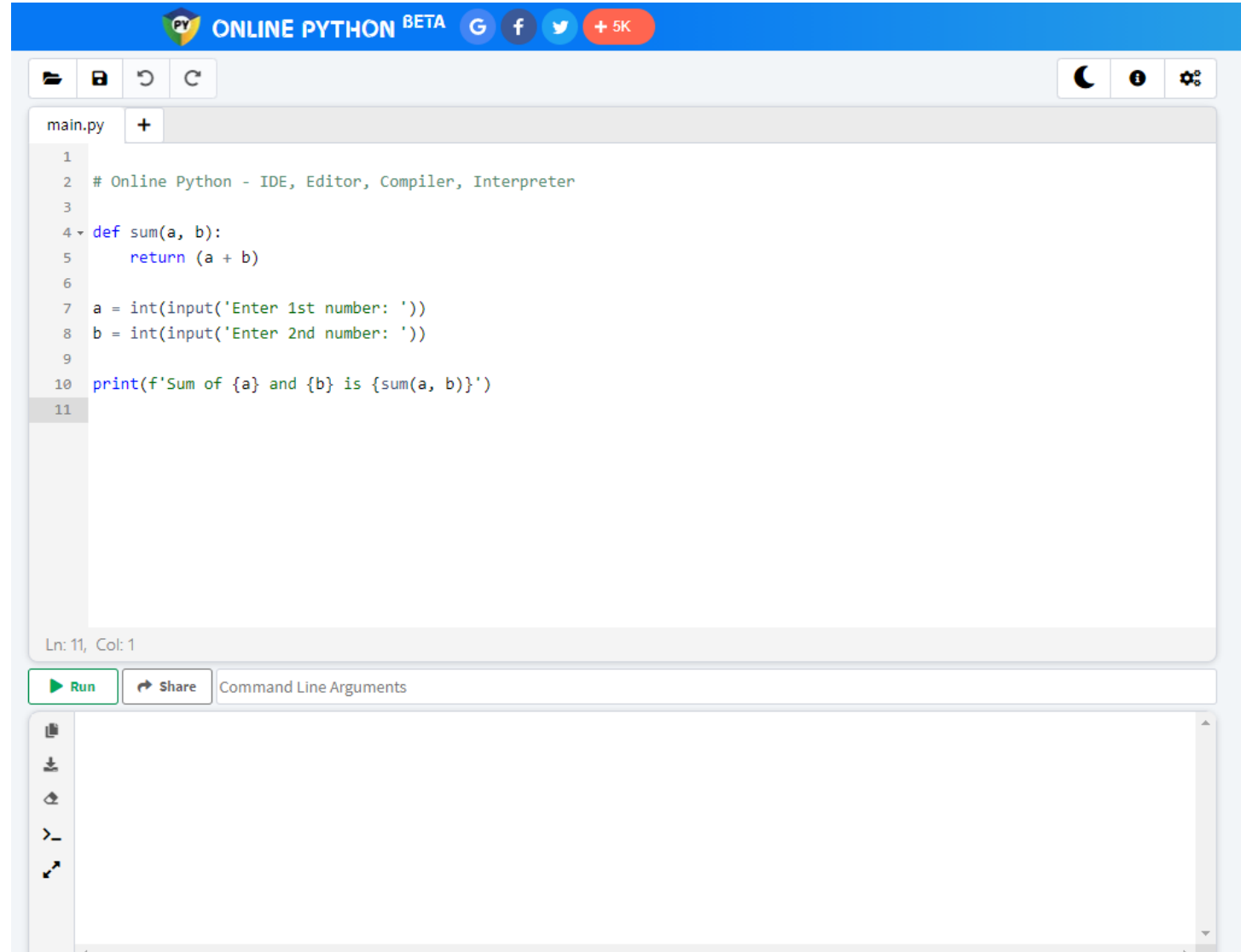
```
hello.py ×  
D: > hello.py > ...  
1  A = 1  
2  B = 2  
3  C = A + B  
4  print(C)  
5  name = "yzu"  
6  print("hello", name)
```

- Run with command
 - python "filename"

```
(base) D:\>python hello.py  
3  
hello yzu
```

Online Tool

- You can utilize online compiler.
- <https://www.online-python.com/>



The screenshot shows the Online Python IDE interface. At the top, there's a blue header with the "ONLINE PYTHON BETA" logo and social media icons (Google, Facebook, Twitter) and a "+ 5K" badge. Below the header is a toolbar with icons for file operations (new, open, save, undo, redo) and a settings icon. The main editor area displays a Python file named "main.py" with the following code:

```
1
2 # Online Python - IDE, Editor, Compiler, Interpreter
3
4 def sum(a, b):
5     return (a + b)
6
7 a = int(input('Enter 1st number: '))
8 b = int(input('Enter 2nd number: '))
9
10 print(f'Sum of {a} and {b} is {sum(a, b)}')
11
```

Below the code editor, there's a status bar showing "Ln: 11, Col: 1". At the bottom, there are buttons for "Run" (with a green play icon) and "Share" (with a share icon), followed by a text input field for "Command Line Arguments". On the left side of the bottom panel, there are icons for file operations (new, open, save, undo, redo) and a settings icon.

Offline Installation

- Python official website
 - <https://www.python.org/downloads/>
- Anaconda
 - <https://www.anaconda.com/products/individual>

Indentation

- Determine the grouping of statements.
- In C++, we use { } to form a block of statements.
- **Leading whitespace (spaces and tabs)** at the beginning of a logical line is used to compute the indentation level of the line.

Built-in Types

- <https://docs.python.org/3/library/stdtypes.html>
- The principal built-in types are numerics, sequences, mappings, classes, instances and exceptions.
 - Built-in Types
 - Truth Value Testing
 - Boolean Operations — and, or, not
 - Comparisons
 - Numeric Types — int, float, complex
 - Iterator Types
 - Sequence Types — list, tuple, range
 - Text Sequence Type — str
 - Binary Sequence Types — bytes, bytearray, memoryview
 - Set Types — set, frozenset
 - Mapping Types — dict
 - Context Manager Types
 - Other Built-in Types
 - Special Attributes

Boolean Operations

Operation	Result	Notes
<code>x or y</code>	if <code>x</code> is false, then <code>y</code> , else <code>x</code>	(1)
<code>x and y</code>	if <code>x</code> is false, then <code>x</code> , else <code>y</code>	(2)
<code>not x</code>	if <code>x</code> is false, then <code>True</code> , else <code>False</code>	(3)

Notes:

1. This is a short-circuit operator, so it only evaluates the second argument if the first one is false.
2. This is a short-circuit operator, so it only evaluates the second argument if the first one is true.
3. `not` has a lower priority than non-Boolean operators, so `not a == b` is interpreted as `not (a == b)`, and `a == not b` is a syntax error.

Comparisons

Operation	Meaning
<	strictly less than
<=	less than or equal
>	strictly greater than
>=	greater than or equal
==	equal
!=	not equal
is	object identity
is not	negated object identity

```
>>> x = 1
>>> x
1
>>> x is 1
True
>>> type(x) is int
True
>>> type(x) is float
False
```


Numeric Types

- int, float, complex
- All numeric types (except complex) support the following operations.

Operation	Result
<code>x + y</code>	sum of <code>x</code> and <code>y</code>
<code>x - y</code>	difference of <code>x</code> and <code>y</code>
<code>x * y</code>	product of <code>x</code> and <code>y</code>
<code>x / y</code>	quotient of <code>x</code> and <code>y</code>
<code>x // y</code>	floored quotient of <code>x</code> and <code>y</code>
<code>x % y</code>	remainder of <code>x / y</code>
<code>-x</code>	<code>x</code> negated
<code>+x</code>	<code>x</code> unchanged
<code>abs(x)</code>	absolute value or magnitude of <code>x</code>
<code>int(x)</code>	<code>x</code> converted to integer
<code>float(x)</code>	<code>x</code> converted to floating point
<code>complex(re, im)</code>	a complex number with real part <code>re</code> , imaginary part <code>im</code> . <code>im</code> defaults to zero.
<code>c.conjugate()</code>	conjugate of the complex number <code>c</code>
<code>divmod(x, y)</code>	the pair <code>(x // y, x % y)</code>
<code>pow(x, y)</code>	<code>x</code> to the power <code>y</code>
<code>x ** y</code>	<code>x</code> to the power <code>y</code>

Bitwise Operations

- Bitwise Operations on Integer Types

Operation	Result
<code>x y</code>	bitwise <i>or</i> of <i>x</i> and <i>y</i>
<code>x ^ y</code>	bitwise <i>exclusive or</i> of <i>x</i> and <i>y</i>
<code>x & y</code>	bitwise <i>and</i> of <i>x</i> and <i>y</i>
<code>x << n</code>	<i>x</i> shifted left by <i>n</i> bits
<code>x >> n</code>	<i>x</i> shifted right by <i>n</i> bits
<code>~x</code>	the bits of <i>x</i> inverted

Arithmetic

```
>>> 3 + 3
6
>>> 60 - 4 * 5
40
>>> (60 - 5*4) / 4
10.0
>>> 9/5
1.8
>>> _
```

The division result is a floating number.

Complex number

```
>>> x = (4 + 5j)
>>> x
(4+5j)
>>> type(x)
<class 'complex'>
```

```
>>> 20 / 3
6.666666666666667
>>> 20 // 3
6
>>> 20 % 3
2
```

```
>>> 2 ** 2
4
>>> 2 ** 10
1024
```

Powers of Numbers

```
>>> PI = 3.14
>>> r = 6
>>> Area = r * r * PI
>>> Area
113.04
```

Sequence Types

- There are three basic sequence types: `lists`, `tuples`, and `range` objects.
- Common Sequence Operations

Operation	Result
<code>x in s</code>	True if an item of <code>s</code> is equal to <code>x</code> , else False
<code>x not in s</code>	False if an item of <code>s</code> is equal to <code>x</code> , else True
<code>s + t</code>	the concatenation of <code>s</code> and <code>t</code>
<code>s * n</code> OR <code>n * s</code>	equivalent to adding <code>s</code> to itself <code>n</code> times
<code>s[i]</code>	<code>i</code> th item of <code>s</code> , origin 0
<code>s[i:j]</code>	slice of <code>s</code> from <code>i</code> to <code>j</code>
<code>s[i:j:k]</code>	slice of <code>s</code> from <code>i</code> to <code>j</code> with step <code>k</code>
<code>len(s)</code>	length of <code>s</code>
<code>min(s)</code>	smallest item of <code>s</code>
<code>max(s)</code>	largest item of <code>s</code>
<code>s.index(x[, i[, j]])</code>	index of the first occurrence of <code>x</code> in <code>s</code> (at or after index <code>i</code> and before index <code>j</code>)
<code>s.count(x)</code>	total number of occurrences of <code>x</code> in <code>s</code>

List (1/2)

- Lists are **mutable sequences**, typically used to store collections of homogeneous items

- A compound data type:

[0]

[2.3, 4.5]

[5, "Hello", "there", 9.8]

[]

- Use [] to index items in the list.
- Use len() to get the length of a list.

```
>>> c = []
>>> c
[]
>>> c.append(2.3)
>>> c.append(4.5)
>>> c
[2.3, 4.5]
>>> c.clear()
>>> c
[]
>>> c.extend([5, "Hello", "there", 9.8])
>>> c
[5, 'Hello', 'there', 9.8]
>>> c[3]
9.8
>>> c = []
>>> c
[]
>>>
```

List (2/2)

- Lists may be constructed in several ways:
 - Using a pair of square brackets to denote the empty list: []
 - Using square brackets, separating items with commas: [a], [a, b, c]
 - Using a list comprehension: [x for x in iterable]
 - Using the type constructor: list() or list(iterable)

Lists are mutable - Some useful methods

```
>>> ids = ["9pti", "2plv", "1crn"]
```

```
>>> ids.append("1alm")
```

```
>>> ids
```

append an element

```
['9pti', '2plv', '1crn', '1alm']
```

```
>>>ids.extend(L)
```

Extend the list by appending all the items in the given list; equivalent to `a[len(a):] = L`.

```
>>> del ids[0]
```

```
>>> ids
```

remove an element

```
['2plv', '1crn', '1alm']
```

```
>>> ids.sort()
```

```
>>> ids
```

sort by default order

```
['1alm', '1crn', '2plv']
```

```
>>> ids.reverse()
```

```
>>> ids
```

reverse the elements in a list

```
['2plv', '1crn', '1alm']
```

```
>>> ids.insert(0, "9pti")
```

```
>>> ids
```

insert an element at some
specified position.
(slower than `.append()`)

```
['9pti', '2plv', '1crn', '1alm']
```

Tuples

- Tuples are **immutable sequences**, typically used to store collections of heterogeneous data
- Tuples may be constructed in a number of ways:
 - Using a pair of parentheses to denote the empty tuple: `()`
 - Using a trailing comma for a singleton tuple: `a`, or `(a,)`
 - Separating items with commas: `a, b, c` or `(a, b, c)`
 - Using the `tuple()` built-in: `tuple()` or `tuple(iterable)`

Tuples: sort of an immutable list

```
>>> yellow = (255, 255, 0) # r, g, b
```

```
>>> one = (1,)
```

```
>>> yellow[0]
```

```
255
```

```
>>> yellow[1:]
```

```
(255, 0)
```

```
>>> yellow[0] = 0
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'tuple' object does not support item assignment

Very common in string interpolation:

```
>>> "%s lives in %s at latitude %.1f" % ("Andrew", "Sweden", 57.7056)
```

```
'Andrew lives in Sweden at latitude 57.7'
```

Ranges

- The range type represents an immutable sequence of numbers and is commonly used for looping a specific number of times in for loops.
 - “range” creates a list of numbers in a specified range
 - range([start,] stop[, step]) -> list of integers
 - When step is given, it specifies the increment (or decrement).

```
>>> range(5)
```

```
[0, 1, 2, 3, 4]
```

```
>>> range(5, 10)
```

```
[5, 6, 7, 8, 9]
```

```
>>> range(0, 10, 2)
```

```
[0, 2, 4, 6, 8]
```

How to get every second element in a list?

```
for i in range(0, len(data), 2):  
    print data[i]
```

Mapping Types

- A mapping object maps hashable values to arbitrary objects. Mappings are mutable objects.
- [Dictionary](#) is a lookup table.
- It maps from a 'key' to a 'value'.

```
>>> studentList = {'s001': "Alice", 's002': "Bob"}
>>> studentList
{'s001': 'Alice', 's002': 'Bob'}
>>> studentList['s001']
'Alice'
>>> studentList['s002']
'Bob'
>>> studentList['s003']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 's003'
```

- Duplicate keys are not allowed.
- Duplicate values are all right.

Keys can be any immutable value
numbers, strings, tuples, frozenset,
not list, dictionary, set, ...

```
atomic_number_to_name = {  
    1: "hydrogen"  
    6: "carbon",  
    7: "nitrogen"  
    8: "oxygen",  
}  
  
nobel_prize_winners = {  
    (1979, "physics"): ["Glashow", "Salam", "Weinberg"],  
    (1962, "chemistry"): ["Hodgkin"],  
    (1984, "biology"): ["McClintock"],  
}
```

A set is an unordered collection
with no duplicate elements.

Dictionary

```
symbol_to_name = {  
    "H": "hydrogen",  
    "He": "helium",  
    "Li": "lithium",  
    "C": "carbon",  
    "O": "oxygen",  
    "N": "nitrogen"  
}
```

```
>>> symbol_to_name["C"]
```

```
'carbon'
```

```
>>> "O" in symbol_to_name, "U" in symbol_to_name
```

```
(True, False)
```

```
>>> "oxygen" in symbol_to_name
```

```
False
```

```
>>> symbol_to_name["P"]
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
KeyError: 'P'
```

```
>>> symbol_to_name.get("P", "unknown")
```

```
'unknown'
```

```
>>> symbol_to_name.get("C", "unknown")
```

```
'carbon'
```

Get the value for a given key

Test if the key exists
("in" only checks the keys,
not the values.)

[] lookup failures raise an exception.
Use ".get()" if you want
to return a default value.

Some useful dictionary methods

```
>>> symbol_to_name.keys()
```

```
['C', 'H', 'O', 'N', 'Li', 'He']
```

```
>>> symbol_to_name.values()
```

```
['carbon', 'hydrogen', 'oxygen', 'nitrogen', 'lithium', 'helium']
```

```
>>> symbol_to_name.update({"P": "phosphorous", "S": "sulfur"})
```

```
>>> symbol_to_name.items()
```

```
[('C', 'carbon'), ('H', 'hydrogen'), ('O', 'oxygen'), ('N', 'nitrogen'), ('P', 'phosphorous'), ('S',  
  'sulfur'), ('Li', 'lithium'), ('He', 'helium')]
```

```
>>> del symbol_to_name['C']
```

```
>>> symbol_to_name
```

```
{'H': 'hydrogen', 'O': 'oxygen', 'N': 'nitrogen', 'Li': 'lithium', 'He': 'helium'}
```

Outline

- Python Basics
- Data Types
- Control Flow
- String
- Files I/O
- Modules
- Function & Class

if

- Condition tests

```
>>> if True:
...     print("True")
...
True
>>> if 1 == 2:
...     print("True")
... else:
...     print("False")
...
False
```

- Multiple condition tests

```
>>> if x == 1:
...     print("1")
... elif x == 2:
...     print("2")
... else:
...     print(">2")
...
>2
```

- Nested conditions

Use “elif” to chain subsequent tests

```
>>> mode = "absolute"
>>> if mode == "canonical":
...     smiles = "canonical"
... elif mode == "isomeric":
...     smiles = "isomeric"
... elif mode == "absolute":
...     smiles = "absolute"
... else:
...     raise TypeError("unknown mode")
...
>>> smiles
'absolute'
>>>
```

“raise” is the Python way to raise exceptions

Boolean Logic

- Python expressions can have “and”s and “or”s:

```
if (ben <= 5 and chen >= 10 or chen == 500 and ben != 5):  
    print “Ben and Chen”
```

Range Test

- Test if the number is in the range.

```
>>> range(10)
range(0, 10)
>>> if 3 in range(10):
...     print("In")
...
In
>>> if 11 in range(10):
...     print("In")
...
>>>
```

For

```
>>> symbols = ["alpha", "beta", "carlie", "delta", "echo", "foxtrot"]
>>> for s in symbols:
...     print(s)
...
alpha
beta
carlie
delta
echo
foxtrot
>>> for i in range(len(symbols)):
...     print(symbols[i])
...
alpha
beta
carlie
delta
echo
foxtrot
```

Tuple assignment in for loops

```
>>> data = [ ("C20H20O3", 308.371),  
             ("C22H20O2", 316.393),  
             ("C24H40N4O2", 416.6),  
             ("C14H25N5O3", 311.38),  
             ("C15H20O2", 232.3181)]
```

```
>>> for (formula, mw) in data:  
...     print "The molecular weight of %s is %s" % (formula, mw)
```

The molecular weight of C20H20O3 is 308.371

The molecular weight of C22H20O2 is 316.393

The molecular weight of C24H40N4O2 is 416.6

The molecular weight of C14H25N5O3 is 311.38

The molecular weight of C15H20O2 is 232.3181

Break, Continue

```
>>> for value in [3, 1, 4, 1, 5, 9, 2]:
...     print ("Checking", value)
...     if value > 8:
...         print ("Exiting for loop")
...         break
...     elif value < 3:
...         print ("Ignoring")
...         continue
...     print ("The square is", value**2)
...
...
```

Use “break” to stop
the for loop

Use “continue” to stop
processing the current item

```
Checking 3
The square is 9
Checking 1
Ignoring
Checking 4
The square is 16
Checking 1
Ignoring
Checking 5
The square is 25
Checking 9
Exiting for loop
>>>
```

While

```
>>> counter = 1
>>> while counter < 10:
...     print(counter)
...     counter = counter + 1
...
1
2
3
4
5
6
7
8
9
```

- Indefinite loop

```
>>> while True:
...     cmd = input("input: ")
...     if cmd == "q":
...         break
...
input: c
input: d
input: f
input: q
```

- “pass” statement does nothing

```
>>> while True:
...     pass
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
KeyboardInterrupt
>>>
```

Outline

- Python Basics
- Data Types
- Control Flow
- String
- Files I/O
- Modules
- Function & Class

Strings share many features with lists

```
>>> smiles = "C(=N)(N)N.C(=O)(O)O"
```

```
>>> smiles[0]
```

```
'C'
```

```
>>> smiles[1]
```

```
(''
```

```
>>> smiles[-1]
```

```
'O'
```


```
>>> smiles[1:5]
```

```
'(=N)'
```

```
>>> smiles[10:-4]
```

```
'C(=O)'
```

Use “slice” notation to
get a substring



String Methods: find, split

```
smiles = "C(=N)(N)N.C(=O)(O)O"
```

```
>>> smiles.find("(O)")
```

```
15
```

```
>>> smiles.find(".")
```

```
9
```

```
>>> smiles.find(".", 10)
```

```
-1
```

```
>>> smiles.split(".")
```

```
['C(=N)(N)N', 'C(=O)(O)O']
```

```
>>>
```

Use “find” to find the start of a substring.

Start looking at position 10.

Find returns -1 if it couldn't find a match.

Split the string into parts with “.” as the delimiter

String operators: in, not in

```
if "Br" in "Brother":
```

```
    print "contains brother"
```

```
email_address = "clin"
```

```
if "@" not in email_address:
```

```
    email_address += "@brandeis.edu"
```

String Method: “strip”, “rstrip”, “lstrip” are ways to remove whitespace or selected characters

```
>>> line = " # This is a comment line \n"
```

```
>>> line.strip()
```

```
'# This is a comment line'
```

```
>>> line.rstrip()
```

```
' # This is a comment line'
```

```
>>> line.rstrip("\n")
```

```
' # This is a comment line '
```

```
>>>
```

More String methods

```
email.startswith("c")  endswith("u")
```

True/False

```
>>> "%s@brandeis.edu" % "clin"
```

'clin@brandeis.edu'

```
>>> names = ["Ben", "Chen", "Yaqin"]
```

```
>>> ", ".join(names)
```

'Ben, Chen, Yaqin'

```
>>> "chen".upper()
```

'CHEN'

Unexpected things about strings

```
>>> s = "andrew"
```

```
>>> s[0] = "A"
```

strings are read only

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'str' object does not support item assignment

```
>>> s = "A" + s[1:]
```

```
>>> s
```

```
'Andrew'
```

“\” is for special characters

\n -> newline

\t -> tab

\\ -> backslash

...

But Windows uses backslash for directories!

filename = "M:\nickel_project\reactive.smi" # DANGER!

filename = "M:\\nickel_project\\reactive.smi" # Better!

filename = "M:/nickel_project/reactive.smi" # Usually works

Outline

- Python Basics
- Data Types
- Control Flow
- String
- Files I/O
- Modules
- Function & Class

Reading files

```
>>> f = open("names.txt")
```

```
>>> f.readline()
```

```
'Yan-Ann Chen\n'
```

Quick Way

```
>>> lst= [ x for x in open("text.txt","r").readlines() ]
```

```
>>> lst
```

```
['Yan-Ann Chen\n', 'chenya@saturn.yzu.edu.tw\n', 'CS348A\n', 'Office Hour: Tues. 7-8, Fri. 8-9\n', 'Yan-Ann Chen\n', 'chenya@saturn.yzu.edu.tw\n', 'CS348B\n', 'Office Hour: Tues. 7-8, Fri. 8-9\n', 'Yan-Ann Chen\n', 'chenya@saturn.yzu.edu.tw\n', 'IN303A\n', 'Office Hour: Tues. 7-8, Fri. 8-9\n']
```

Ignore the header?

```
for (i,line) in enumerate(open('text.txt','r').readlines()):
```

```
    if i == 0: continue
```

```
    print(line)
```

Using dictionaries to count occurrences

```
>>> name_count = {}  
>>> for line in open('names.txt'):  
...     name = line.strip()  
...     name_count[name] = name_count.get(name,0)+ 1  
...  
>>> for (name, count) in name_count.items():  
...     print(name, count)  
...
```

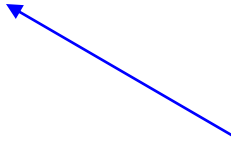
Yan-Ann Chen 3

Chen 3

Yan-Ann 3

File Output

```
input_file = open("in.txt")  
output_file = open("out.txt", "w")  
for line in input_file:  
    output_file.write(line)
```



“w” = “write mode”
“a” = “append mode”
“wb” = “write in binary”
“r” = “read mode” (default)
“rb” = “read in binary”
“U” = “read files with Unix
or Windows line endings”

Outline

- Python Basics
- Data Types
- Control Flow
- String
- Files I/O
- Modules
- Function & Class

Modules

- When a Python program starts it only has access to a basic functions and classes.

(“int”, “dict”, “len”, “sum”, “range”, ...)

- “Modules” contain additional functionality.
- Use “import” to tell Python to load a module.

```
>>> import math
```

```
>>> import nltk
```

import the math module

```
>>> import math
```

```
>>> math.pi
```

```
3.1415926535897931
```

```
>>> math.cos(0)
```

```
1.0
```

```
>>> math.cos(math.pi)
```

```
-1.0
```

```
>>> dir(math)
```

```
['__doc__', '__file__', '__name__', '__package__', 'acos', 'acosh',  
'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos',  
'cosh', 'degrees', 'e', 'exp', 'fabs', 'factorial', 'floor', 'fmod',  
'frexp', 'fsum', 'hypot', 'isinf', 'isnan', 'ldexp', 'log', 'log10',  
'log1p', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan',  
'tanh', 'trunc']
```

```
>>> help(math)
```

```
>>> help(math.cos)
```

“import” and “from ... import ...”

```
>>> import math
```

```
math.cos
```

```
>>> from math import cos, pi
```

```
cos
```

```
>>> from math import *
```


Outline

- Python Basics
- Data Types
- Control Flow
- String
- Files I/O
- Modules
- Function & Class

Function

```
def function_name():  
    process  
  
def function_name(param_name):  
    process  
  
def function_name(param_name = 3):  
    process
```

- Declare “global” if you want to modify a global variable in a function.

Classes

```
class ClassName(object):
```

```
    <statement-1>
```

```
    ...
```

```
    <statement-N>
```

```
class MyClass(object):
```

```
    """A simple example class"""
```

```
    i = 12345
```

```
    def f(self):
```

```
        return self.i
```

```
class DerivedClassName(BaseClassName):
```

```
    <statement-1>
```

```
    ...
```

```
    <statement-N>
```

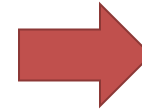
Lab

- Download “gen.py”.
 - <https://reurl.cc/ERD86n>
- Run “gen.py” to get a “symbols.txt”
- Input the mapping
ex: *:1,#:2,@:3,\$:4
- Read the content of “symbols.txt” in your program.
- Map each symbol to its corresponding number and output.
(You cannot directly output the result.)
- Hints
 - print(xxx, end=""): print without newline
 - input(): get and return the input data
 - str.split(): get substrings without the delimiter.

```

1  **@#@#@#@#@#$#$#$#@#*##
2  @*@
3  @$#@###@$**@#*#@#@#@#@
4  *@*$@*$#
5  **#@#@#@#@#$
6  @
7  $@@*$#$#@###@$*#$*#@##
8  #$$$*#@#@*
9  *@@$###*@@@
10 @#*@$**###@#$*#@##
11 $#$@**#$**$###
12 @*#####@#*#$*@###*#@*##
13 *@@**$#@#*#@*#$#$#@**$#$#$#
14 *@$*$**$###$#@*@@@*#$**#@@@$@@
15 @*$*$*$*@$###*#@*$$$*#$#@
16 **$*$#$#
17 *$*@$*$*#@#@#*@$@###$#@
18 @@@$@#@**#@###$*@$#
19 $$$@*$@@$@*#@*##@@$
20 @$#$*#@**#@#$*$*###@###*@

```



```

Input: *:1,#:2,@:3,$:4
{'*': '1', '#': '2', '@': '3', '$': '4'}
113323232334314444323212122
313
34232223411134213313232323
13143142
1123432234314
3
4331444322431243413122
24441313311
133422113313
321342112223412321
42431124114224
3122221321324132221313121
133211423213132442341124341442
134412144224231333141411233433
314144134242231313114423
11414424
141314123321343222243
3343231123224411344
443413334313212334
342413112324141222322213

```

```

print('Enter your name:')
x = input()
print('Hello, ' + x)

```