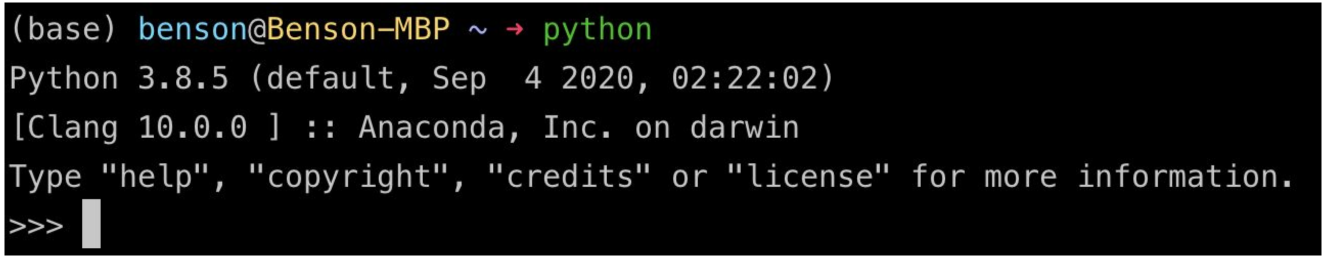
python

R10922192 許雅晴

python introduction

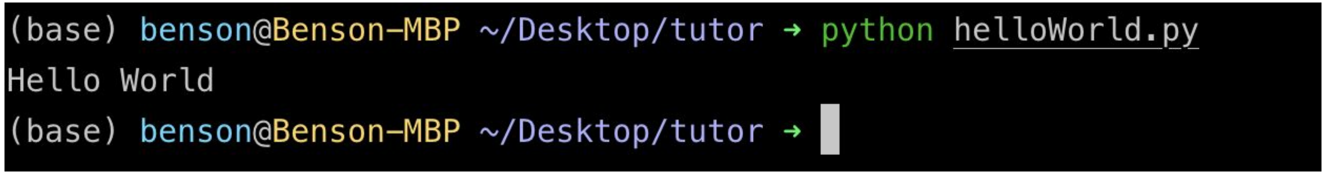
How to execute python?

- Interactive mode
  - Enter your terminal & execute python



```
(base) benson@Benson-MBP ~ ➜ python
Python 3.8.5 (default, Sep  4 2020, 02:22:02)
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

- Script mode
  - python [filename]



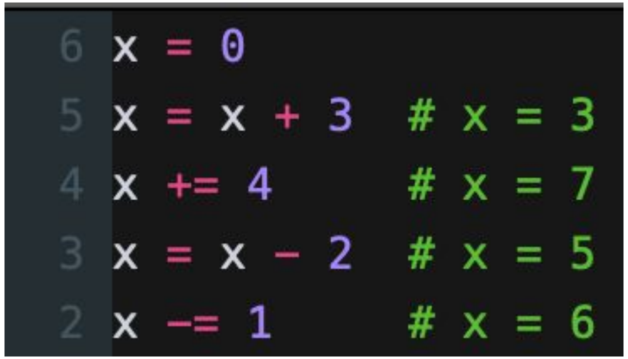
```
(base) benson@Benson-MBP ~/Desktop/tutor ➜ python helloWorld.py
Hello World
(base) benson@Benson-MBP ~/Desktop/tutor ➜ 
```

Built-in data types

Basic knowledge

Variable

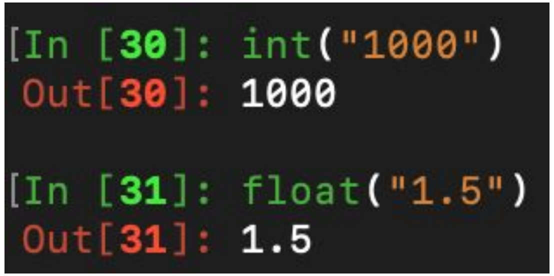
- Variables are containers for storing data values.
- Unlike c/c++, no need to declare type
- Operator:
  - / : divide
  - // : floor (= "/" in c/c++)
  - ** : power
    - e.g. a ** 2: square of a
- Assign value:
  - Same in c/c++
- Basic types:
  - int, float, bool, string, None, ...



```
x = 0
x = x + 3   # x = 3
x += 4      # x = 7
x = x - 2   # x = 5
x -= 1      # x = 6
```

Int, Float

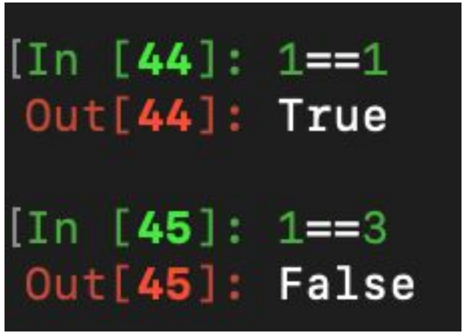
- Float stands for a floating point.
- Int stands for integer.
- You can cast string into int or float.



```
[In [30]: int("1000")
 Out[30]: 1000

[In [31]: float("1.5")
 Out[31]: 1.5
```

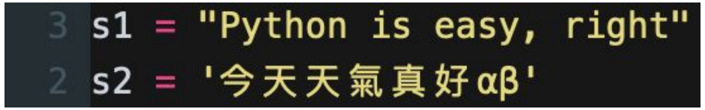
Bool

- Booleans represent one of two values: `True` or `False`
    - compare two values, the expression is evaluated and Python returns the Boolean answer



```
[In [44]: 1==1
 Out[44]: True

[In [45]: 1==3
 Out[45]: False
```

String



```
s1 = "Python is easy, right"
s2 = '今天天氣真好αβ'
```

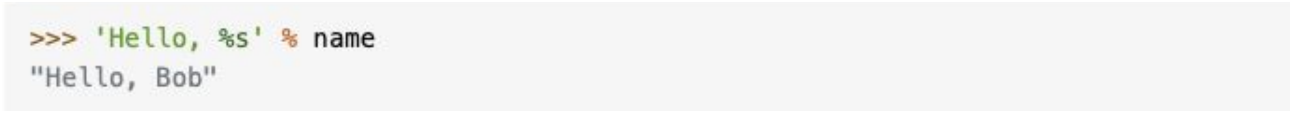
- You can use either ‘ ’ or “ ”
- Operator:
  - Add: “Pyt” + “hon” = “Python”
  - Multiply: “a” * 5 = “aaaaa”
  - in:
    - “cd” in “abcde” == True
    - “ff” in “abcdef” == False
- Other methods:
  - len(str): length of string
  - str.split(): split any whitespace. You can specify separator by putting it into brackets
  - More methods, referring to: https://www.w3schools.com/python/python_ref_string.asp

String

- Get i-th char: str[i]
  - i can be negative integer, meaning that count from back
  - e.g. s = “abcdefghi”
    - s[2] = “c”
    - s[-1] = “i”
- Get substring: str[i:j]
  - Get substring of [i, j) (左閉右開)
  - If it’s from start/end, you don’t need to write it
  - e.g. s = “abcdefghi”
    - s[:2] = s[0:2] = “ab”
    - s[5:] = “fghi”
    - s[5:-2] = “fg”

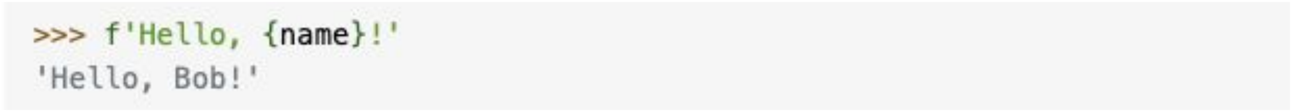
Format String

- Old method:

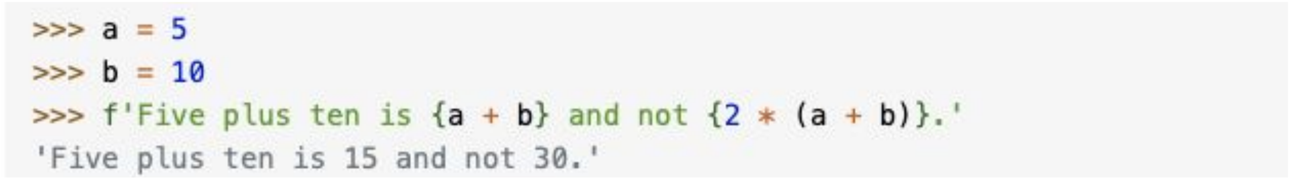


```
>>> 'Hello, %s' % name
"Hello, Bob"
```

- Modern method: f-string



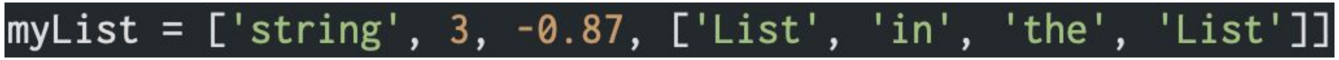
```
>>> f'Hello, {name}!'
'Hello, Bob!'
```



```
>>> a = 5
>>> b = 10
>>> f'Five plus ten is {a + b} and not {2 * (a + b)}.'
'Five plus ten is 15 and not 30.'
```

  - Note: your python environment should >=3.6

List



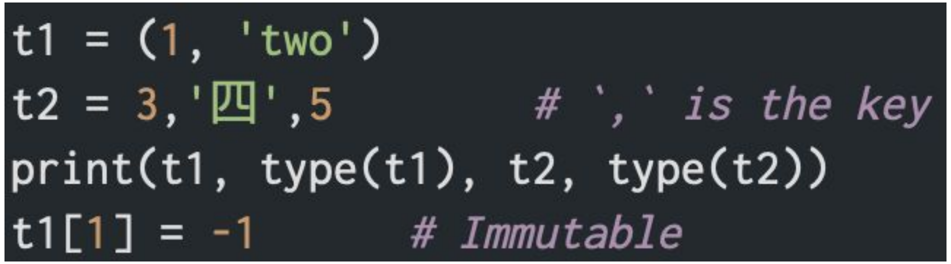
```
myList = ['string', 3, -0.87, ['List', 'in', 'the', 'List']]
```

- you can put items of any type into the list
- Initialization
  - myList = []
  - myList = list()
- Change value:
  - myList[i] = “new thing”
- Operator:
  - Add: [“This”, “is”] + [“a”, “new”, “list”] = [“This”, “is”, “a”, “new”, “list”]
  - Multiply: [“OAO”] * 2 = [“OAO”, “OAO”]

List

- Other methods:
  - Append: myList.append("OAO")
  - Extend: myList.extend(["This", "is", "a", "new", "list"])
  - Length: len(myList)
- Get item & sub-list:
  - Similar to string
  - myList = [1, 2, 3, 4, 5]
    - myList[-1] = 5
    - myList[:2] = [1, 2]

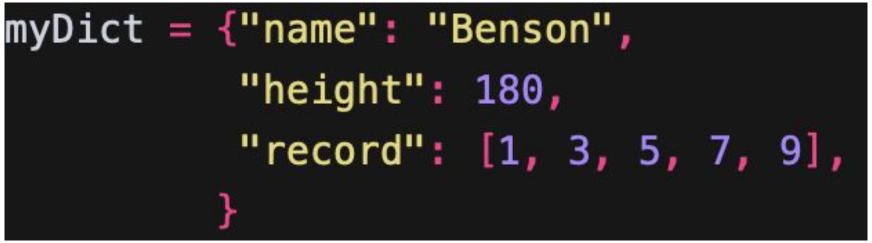
Tuple



```
t1 = (1, 'two')
t2 = 3,'四',5          # `,` is the key
print(t1, type(t1), t2, type(t2))
t1[1] = -1          # Immutable
```

- It’s **immutable**
- len(myTuple): get length of tuple
- Useful tips:
    - you can set multiple variables at the same time by tuple
        - a, b = “TA”, 2
    - swap:
        - a, b = b, a

Dictionary



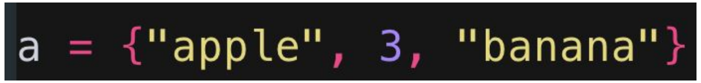
```
myDict = {"name": "Benson",
          "height": 180,
          "record": [1, 3, 5, 7, 9],
         }
```

- Implemented by hash table
  - If you don’t know, learning it in DSA
- pairs of (key, value)
  - a key will have its own value
  - key: can be string or int (string is recommended)
  - value: any type of object you want

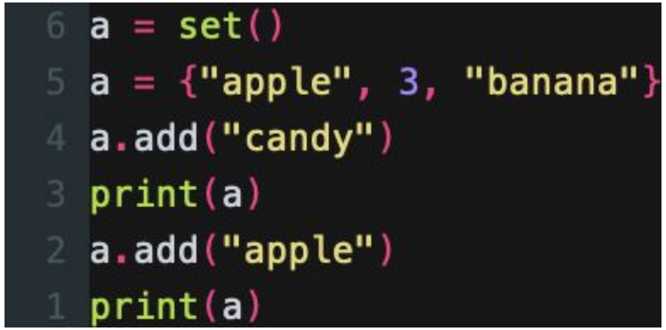
Dictionary

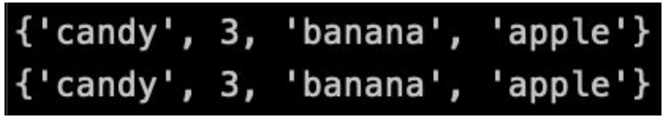
- insert & modify
  - myDict[“name”] = “New Name”
- get content of dict
  - d.keys() : get all keys in d
  - d.values() : get all values in d
  - d.items() : get all (key, value) pairs in d
- Update:
  - dictA.update(dictB)
  - dictA |= dictB
    - Note: Note: your python environment should >= 3.9
    - merges **dictA** with **dictB** and returns the updated **dictA**

Set


```python
a = {"apple", 3, "banana"}
```

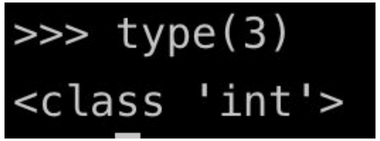
- Similar to dictionary, using { }
  - However, it only contains value
  - Sets are used to store multiple items
- Methods
  - mySet.add("a"): put "a" into mySet
  - check whether an item is in set: in
    - "a" in mySet


```python
a = set()
a = {"apple", 3, "banana"}
a.add("candy")
print(a)
a.add("apple")
print(a)
```


```
{'candy', 3, 'banana', 'apple'}
{'candy', 3, 'banana', 'apple'}
```

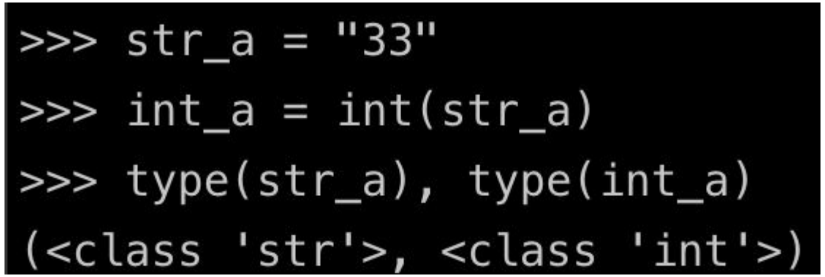
Other useful information

- type(a):
  - get the type of a



```
>>> type(3)
<class 'int'>
```

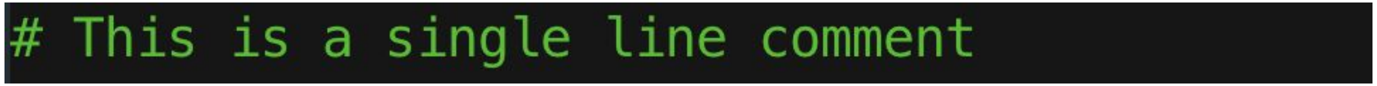
- Change type
  - for example, we want to change the type from str to int



```
>>> str_a = "33"
>>> int_a = int(str_a)
>>> type(str_a), type(int_a)
(<class 'str'>, <class 'int'>)
```

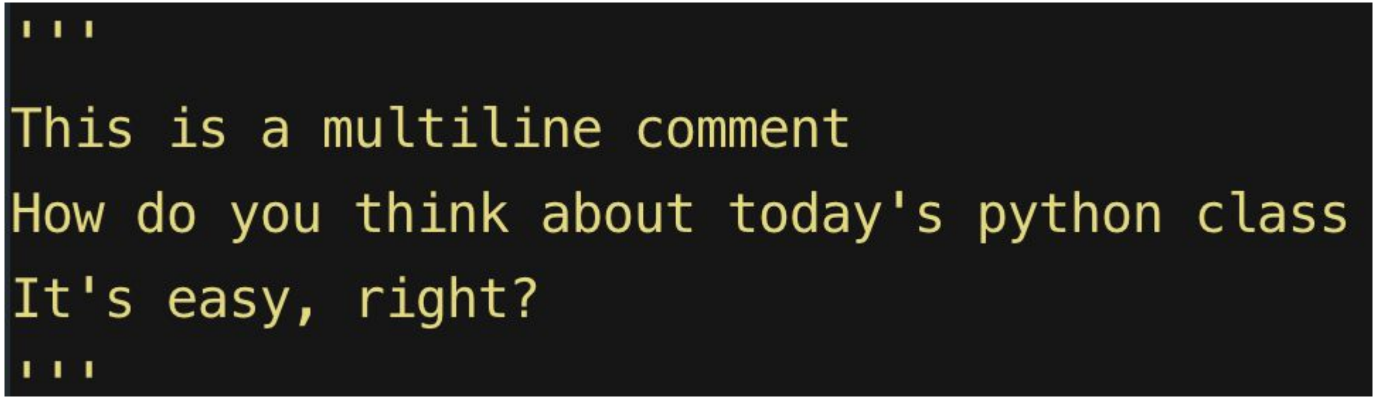
Other useful information

- Single-line comment:



```python
# This is a single line comment
```

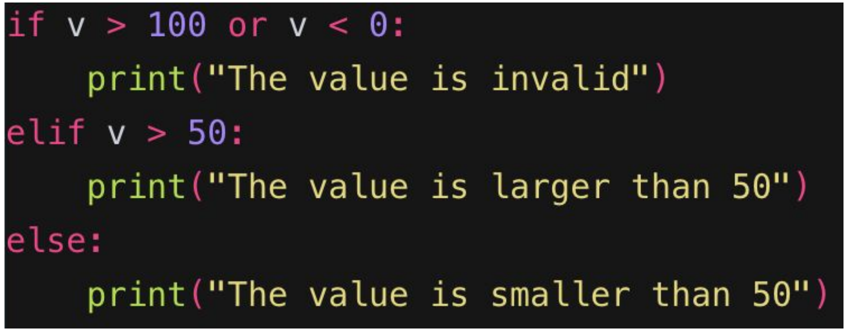
- Multi-line comment:



```python
'''
This is a multiline comment
How do you think about today's python class
It's easy, right?
'''
```

Flow Control

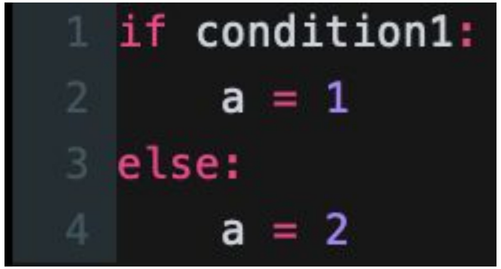
if / elif / else



```
if v > 100 or v < 0:
    print("The value is invalid")
elif v > 50:
    print("The value is larger than 50")
else:
    print("The value is smaller than 50")
```

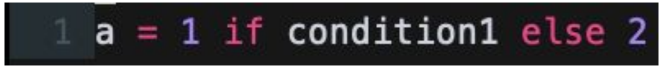
- Difference from c/c++ :
  - No need ( ) and { }, using indent
  - elif means “else if“
- logic operator:
  - not: !
  - and: &&
  - or: | |

Ternary operator (三元運算子)



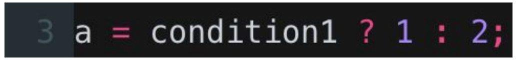
```
if condition1:
    a = 1
else:
    a = 2
```

- Can we code it into 1 line?
- Ternary operator:



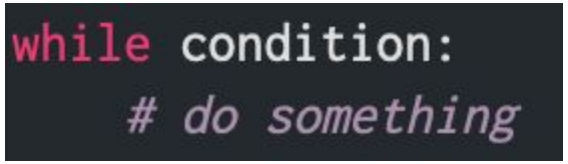
```
a = 1 if condition1 else 2
```

    - Basically, it equals to the following code in c/c++;



```
a = condition1 ? 1 : 2;
```

While loop



```
while condition:
    # do something
```

- while the condition is true, do the following things
- It’s quite similar to what you do in c/c++

For loop

```python
for variable in iterableObject:
    # do something
```

- What is iterable object?
  - object that can be iterated upon, meaning that you can **traverse through all the values**
  - List, tuple: traverse all values in order
  - Set: traverse all values. Since it’s unordered, we can’t ensure the order
  - Dictionary:
    - it’s also unordered
    - iterate **key**
    - if we want to iterate key & value:

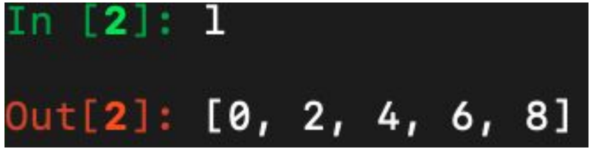
```python
d = {"a": "Apple", "b": "Banana", "c": "Candy"}
for k, v in d.items():
    print(k, v)
```

range()

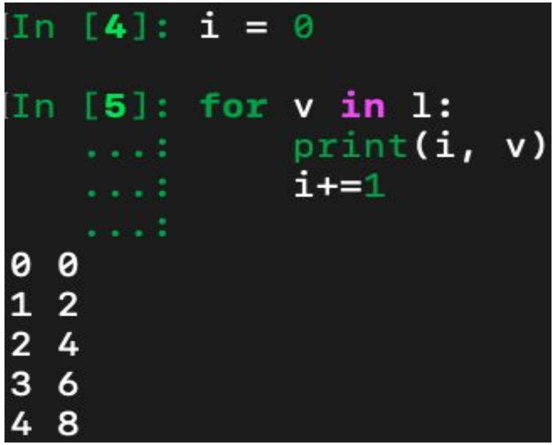
- range(n) : 0, 1, …, n-1
- range(m, n) : m, m+1, …, n-1
- range(m, n, k) : m, m+k, m+ 2k, … until it’s out of range
    - k can be negative -> decrease

For loop & enumerate

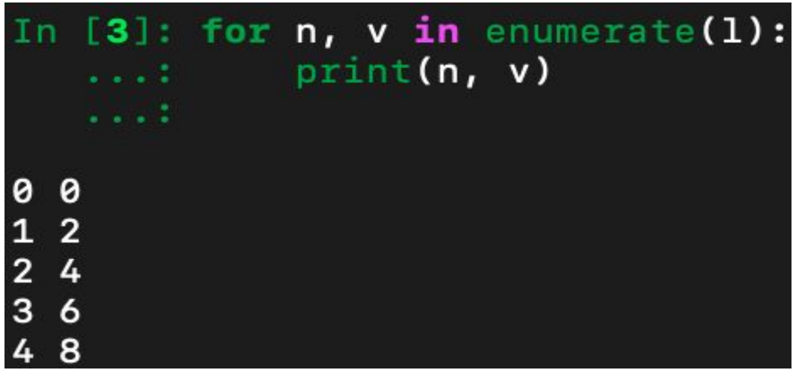
- Enumerate() method adds a counter to an iterable and returns it in a form of enumerating object.



```
In [2]: l

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



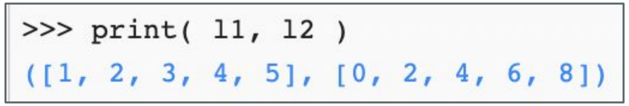
```
In [4]: i = 0

In [5]: for v in l:
   ...:     print(i, v)
   ...:     i+=1
   ...:
0 0
1 2
2 4
3 6
4 8
```



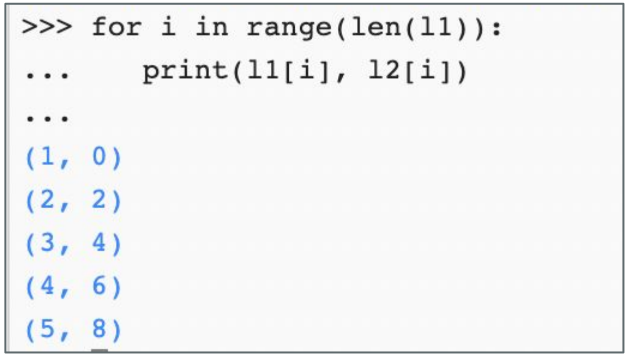
```
In [3]: for n, v in enumerate(l):
   ...:     print(n, v)
   ...:

0 0
1 2
2 4
3 6
4 8
```

For loop & zip

- zip() returns an iterator of tuples with each tuple having elements from all the iterables.

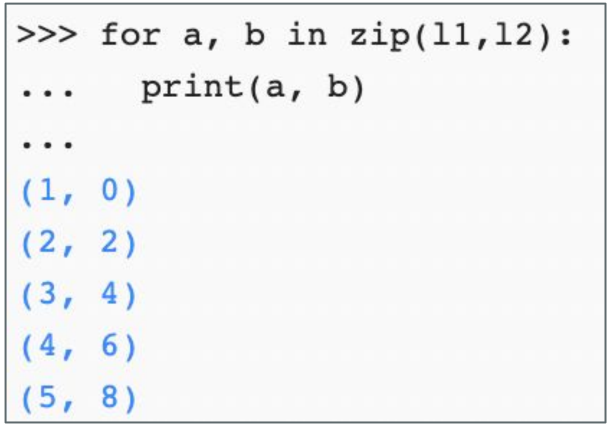


```
>>> print( l1, l2 )
([1, 2, 3, 4, 5], [0, 2, 4, 6, 8])
```



```
>>> for i in range(len(l1)):
...     print(l1[i], l2[i])
...
(1, 0)
(2, 2)
(3, 4)
(4, 6)
(5, 8)
```

Same



```
>>> for a, b in zip(l1,l2):
...     print(a, b)
...
(1, 0)
(2, 2)
(3, 4)
(4, 6)
(5, 8)
```

List comprehension

- Given a list of string, you need to remove any trailing characters of “!”
- List comprehension!!!

```
newlist = [expression for item in iterable if condition == True]
```

  - If you think it’s complicated, we can ignore “if” first
  - In short, it just put for loop into list

List comprehension

```
newlist = [expression for item in iterable if condition == True]
```

- For example:

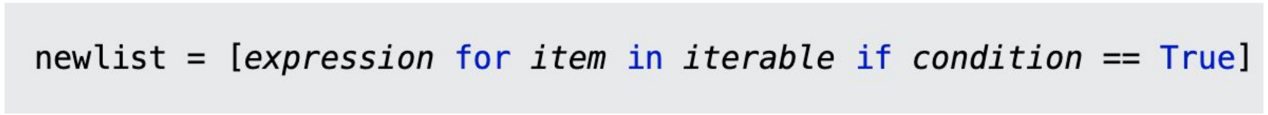
```
begin_sentences = [s.rstrip("!") for s in begin_sentences]
end_sentences = [s.rstrip("!") for s in end_sentences]
```

- Put more than 2 for loop ...

```
>>> s = ["A", "B", "C"]
>>> [i + str(j) for i in s for j in range(3)]
```

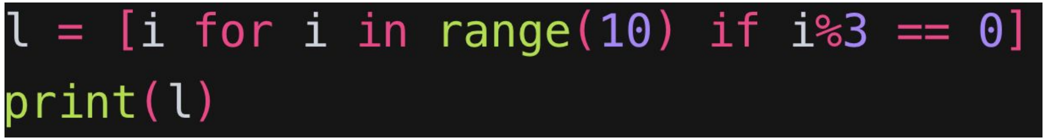
List comprehension

- Let’s put the “if” back

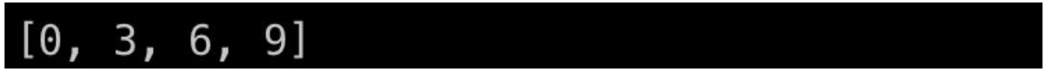


```
newlist = [expression for item in iterable if condition == True]
```

- For example

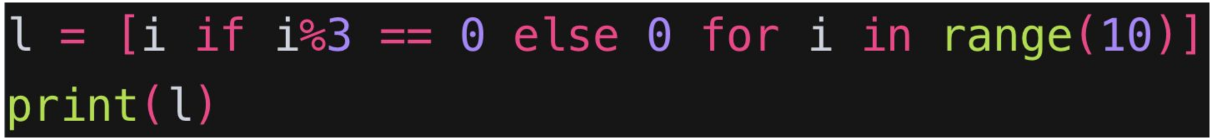


```
l = [i for i in range(10) if i%3 == 0]
print(l)
```



```
[0, 3, 6, 9]
```

- If you also want to use else … ?



```
l = [i if i%3 == 0 else 0 for i in range(10)]
print(l)
```

Functions & Class

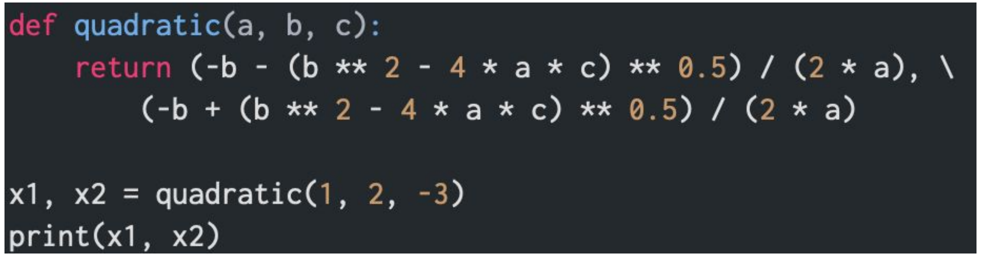
Function

- Basic prototype:

```python
def my_function(arg1, arg2):
    # do what you want here

my_function(arg1, arg2)
```

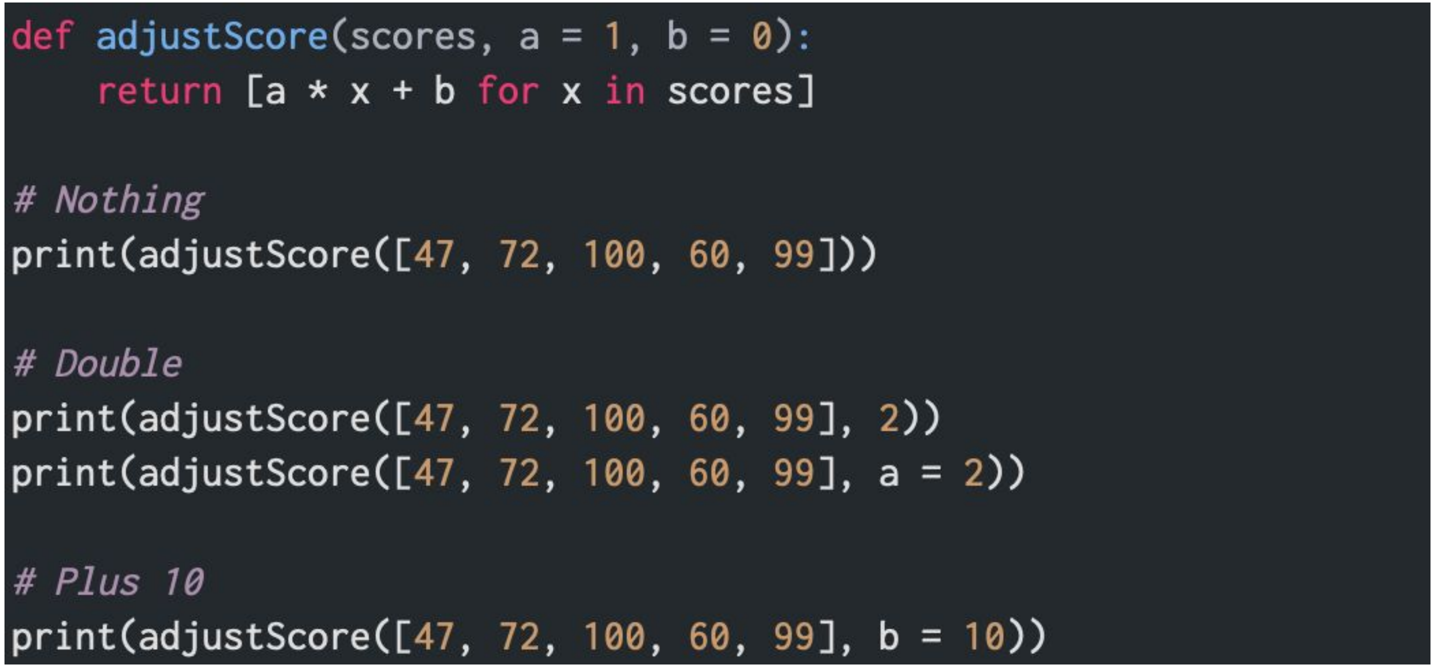
```python
def my_function(arg1, arg2):
    # do what you want here

    return my_return # return something you want

ret = my_function(arg1, arg2)
```

- Unlike c/c++, no need to say what’s your return type
  - if no return anything, the return is None
- return more than 1 (?
  - it just like you return a tuple



```python
def quadratic(a, b, c):
    return (-b - (b ** 2 - 4 * a * c) ** 0.5) / (2 * a), \
        (-b + (b ** 2 - 4 * a * c) ** 0.5) / (2 * a)

x1, x2 = quadratic(1, 2, -3)
print(x1, x2)
```

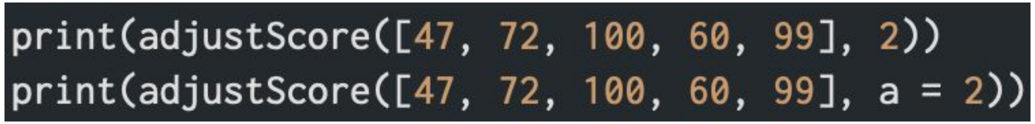
Default parameters value



```python
def adjustScore(scores, a = 1, b = 0):
    return [a * x + b for x in scores]

# Nothing
print(adjustScore([47, 72, 100, 60, 99]))

# Double
print(adjustScore([47, 72, 100, 60, 99], 2))
print(adjustScore([47, 72, 100, 60, 99], a = 2))

# Plus 10
print(adjustScore([47, 72, 100, 60, 99], b = 10))
```

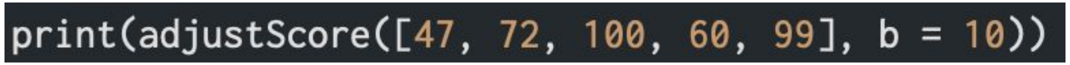
Default parameters value

- Some notice :
  - Arguments w/ default value must put after arguments w/o default value
  - when calling the function
    - Put it in order: you can write “variable=” or not



```
print(adjustScore([47, 72, 100, 60, 99], 2))
print(adjustScore([47, 72, 100, 60, 99], a = 2))
```

    - If ignore passing some arguments, you must write “variable=”

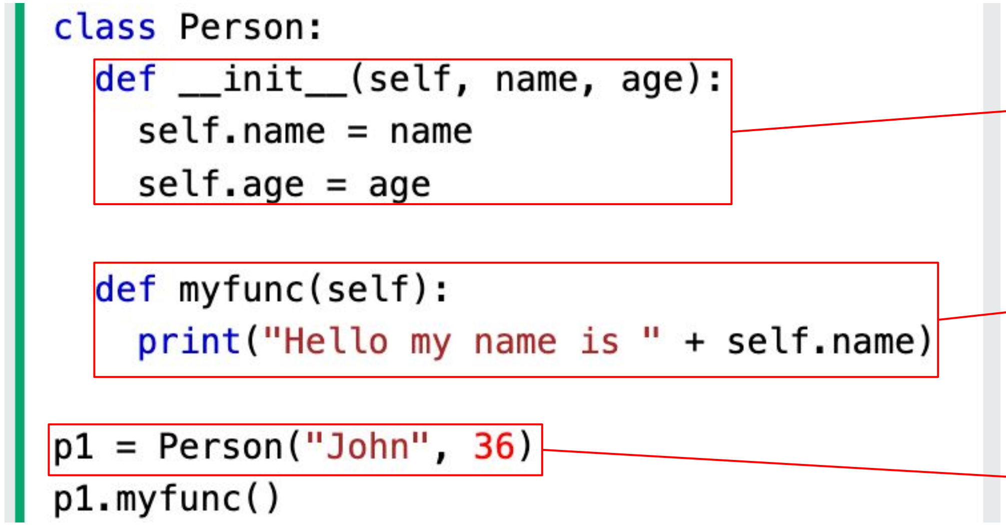


```
print(adjustScore([47, 72, 100, 60, 99], b = 10))
```

    - Once you write “variable=”, you have to write it for all variables after it

adjustScore(score=[10, 50, 100], **a=2**)

Class - basic



```python
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

Constructor:

- Executed when create a new object
- You must put self as first argument

Method: (instance method)

Ttreat it as a function

- You must put self as first argument

Create a new object

Class - inheritance

```python
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)
```

```python
class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

x = Student("Mike", "Olsen", 2019)
```

Inheritance (from “Person”)

- All classes in Python are inherit from object

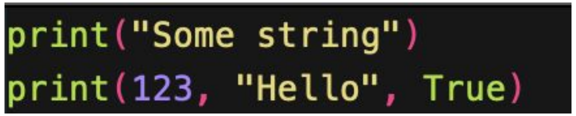
Call contructor of its parent

- super(): get its parent class

IO / formatting

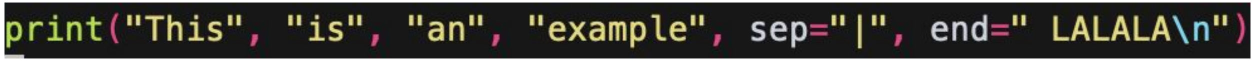
Standard I/O

- Input: input( )
  - return type: string
  - You can put some string as hint in the ( )
    - E.g. name = input(“Please input your name: “)
- Output: print( )
  - You can put any type you want (if it can be printed)



```
print("Some string")
print(123, "Hello", True)
```

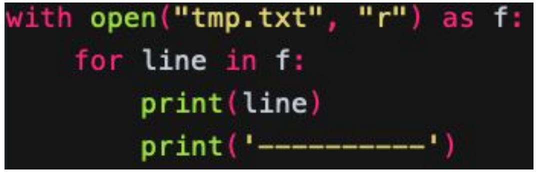
  - You can use “sep=” and “end=” to change separate string & end string



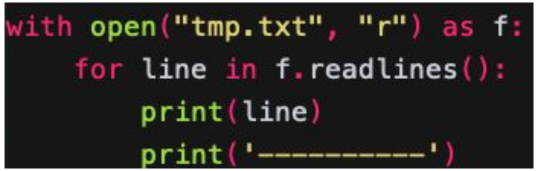
```
print("This", "is", "an", "example", sep="|", end=" LALALA\n")
```

Read from file

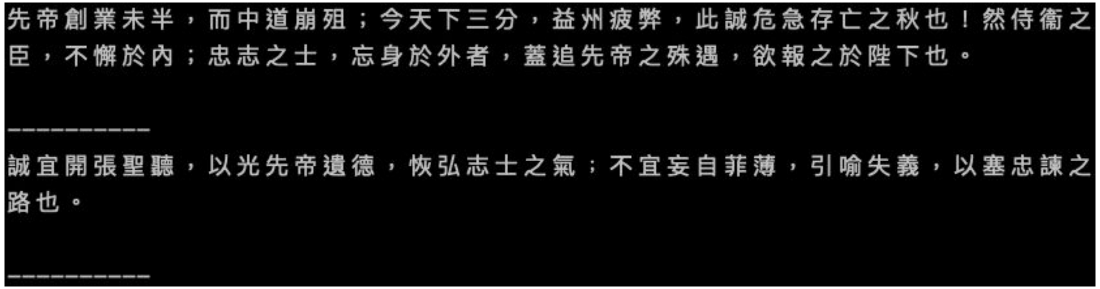
- The below 2 methods can both read all lines in a file:



```python
with open("tmp.txt", "r") as f:
    for line in f:
        print(line)
        print('----------')
```



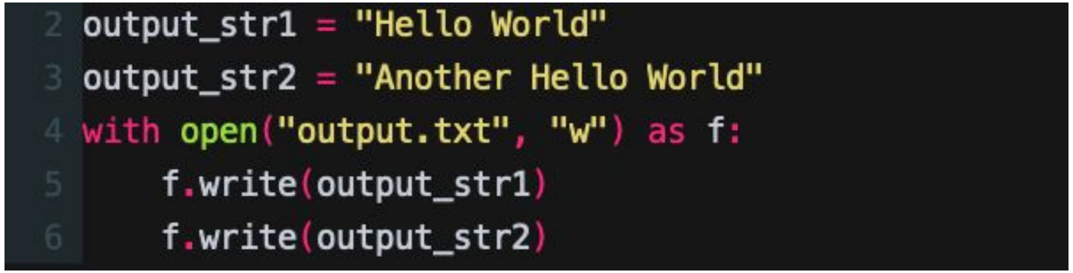
```python
with open("tmp.txt", "r") as f:
    for line in f.readlines():
        print(line)
        print('----------')
```



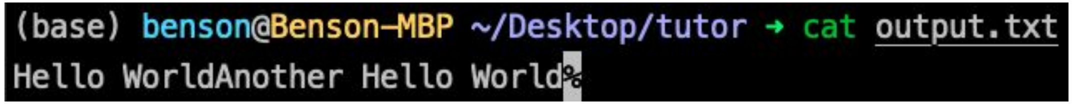
```
先帝創業未半，而中道崩殂；今天下三分，益州疲弊，此誠危急存亡之秋也！然侍衛之
臣，不懈於內；忠志之士，忘身於外者，蓋追先帝之殊遇，欲報之於陛下也。

----------
誠宜開張聖聽，以光先帝遺德，恢弘志士之氣；不宜妄自菲薄，引喻失義，以塞忠諫之
路也。

----------
```

- You can just simply use: f = open(“tmp.txt”, “r”)
  - But you have to close the file pointer (fp) manually
  - f.close()

Write to file



```
output_str1 = "Hello World"
output_str2 = "Another Hello World"
with open("output.txt", "w") as f:
    f.write(output_str1)
    f.write(output_str2)
```

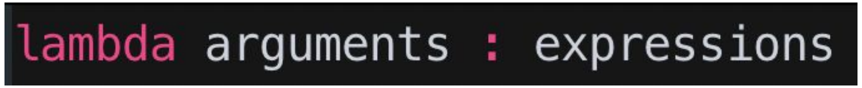


```
(base) benson@Benson-MBP ~/Desktop/tutor ➜ cat output.txt
Hello WorldAnother Hello World%
```

- Samely, you can use f = open(...), but you need to close the fp manually.

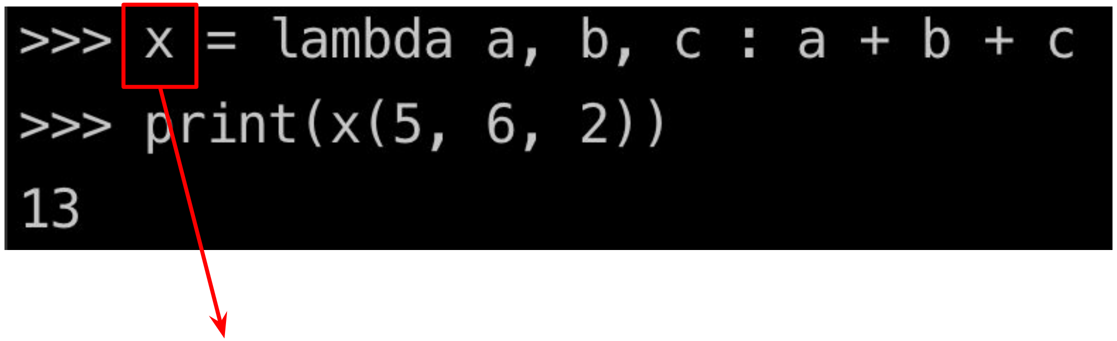
Lambda, map and filter

Lambda



```
lambda arguments : expressions
```

You can think lambda is a simplified version of function.

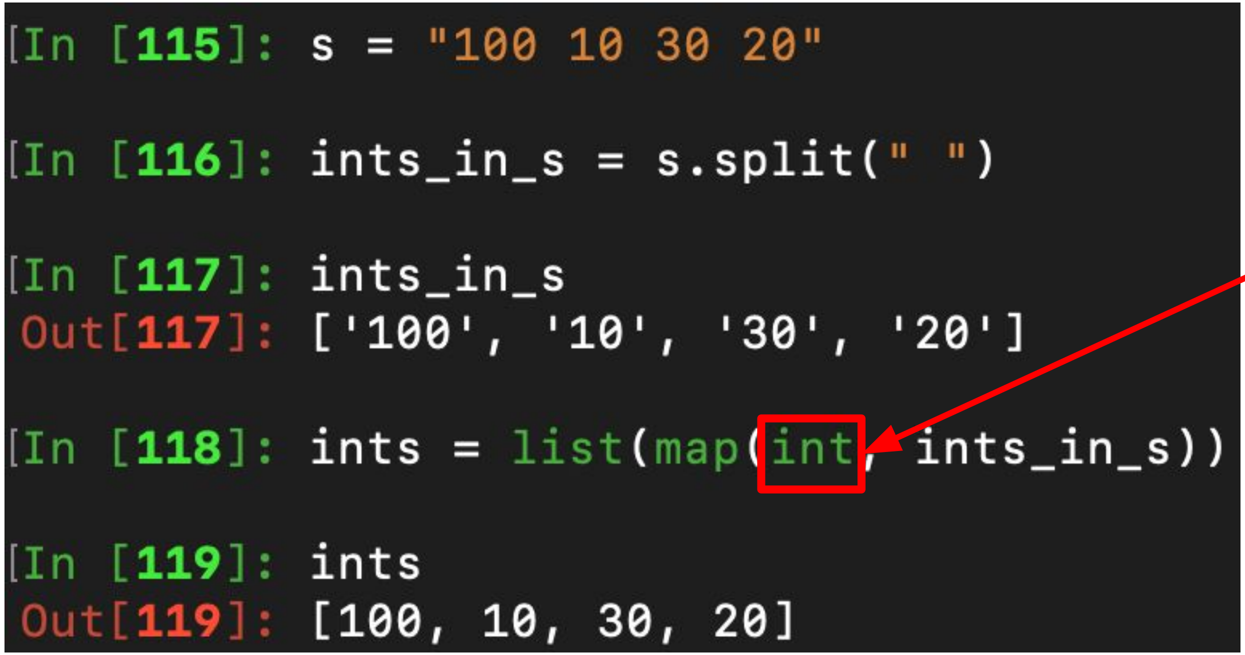


```
>>> x = lambda a, b, c : a + b + c
>>> print(x(5, 6, 2))
13
```

x is a **lambda function**

Map

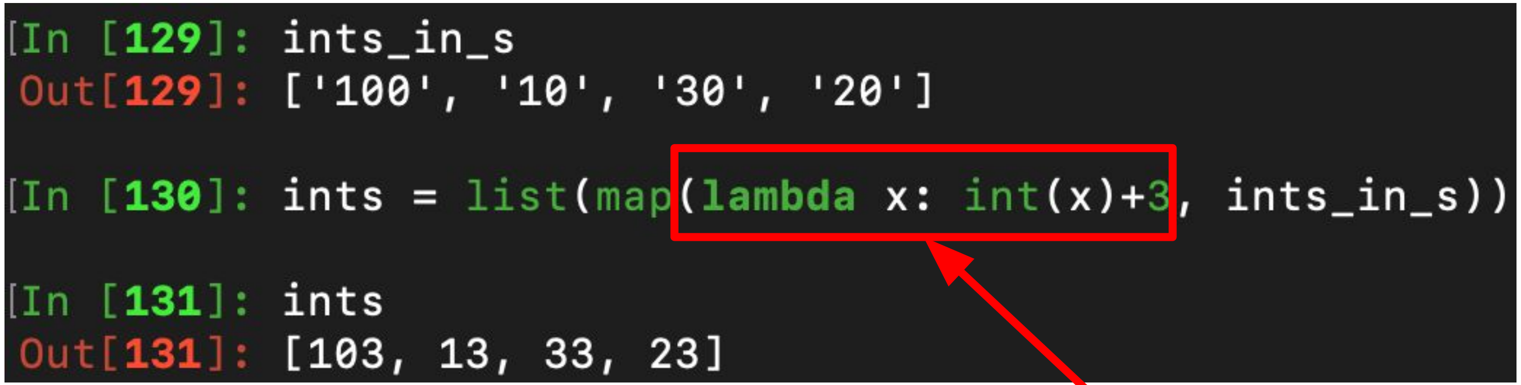
You can use map to simplify your code. Especially the loop parts.



```
[In [115]: s = "100 10 30 20"

[In [116]: ints_in_s = s.split(" ")

[In [117]: ints_in_s
 Out[117]: ['100', '10', '30', '20']

[In [118]: ints = list(map(int, ints_in_s))

[In [119]: ints
 Out[119]: [100, 10, 30, 20]
```

This function will be applied to every element in list

Map, combined with lambda

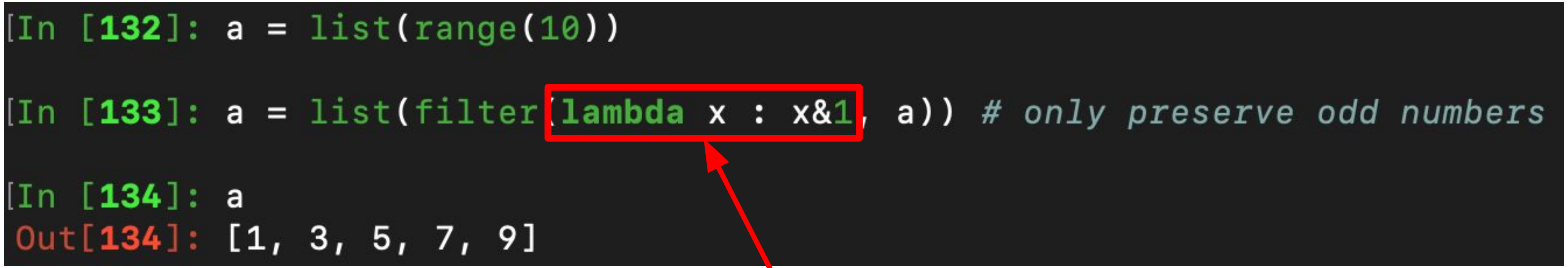


```
[In [129]: ints_in_s
 Out[129]: ['100', '10', '30', '20']

[In [130]: ints = list(map(lambda x: int(x)+3, ints_in_s))

[In [131]: ints
 Out[131]: [103, 13, 33, 23]
```

Convert to int and then +3

Filter

To filter out the element you want to discard.

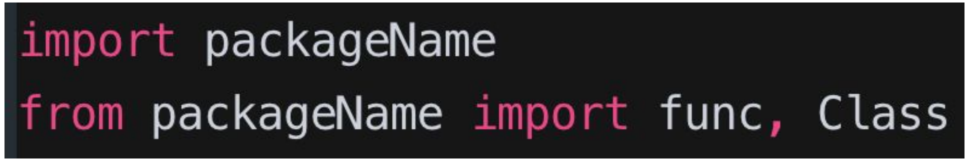


```
[In [132]: a = list(range(10))

[In [133]: a = list(filter(lambda x : x&1, a)) # only preserve odd numbers

[In [134]: a
 Out[134]: [1, 3, 5, 7, 9]
```

- True for preserving
- False for descarding

Python Package

Use packages

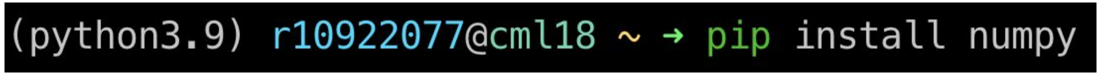


```python
import packageName
from packageName import func, Class
```

- Python standard library:
  - re: regular expression operations
  - pathlib: object-oriented filesystem paths
  - sys: system-specific parameters and functions
  - os: miscellaneous operating system interfaces
  - argparse: parser for command-line options, arguments and sub-commands
  - …
  - Others can refer to: https://docs.python.org/3/library/

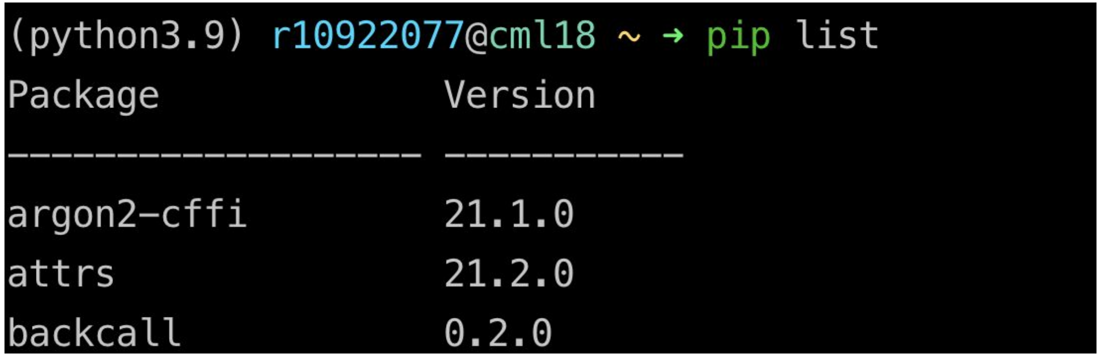
Use other packages

- Use pip (pip3) to achieve that



```
(python3.9) r10922077@cml18 ~ ➜ pip install numpy
```

- If you don’t have pip:
    - curl -sSL https://bootstrap.pypa.io/get-pip.py -o get-pip.py
    - python get-pip.py
- You can use “pip list“ to check what packages you have



```
(python3.9) r10922077@cml18 ~ ➜ pip list
Package             Version
------------------- -----------
argon2-cffi         21.1.0
attrs               21.2.0
backcall            0.2.0
```