Review of Python Basics (version 3.x)

Liang Liang

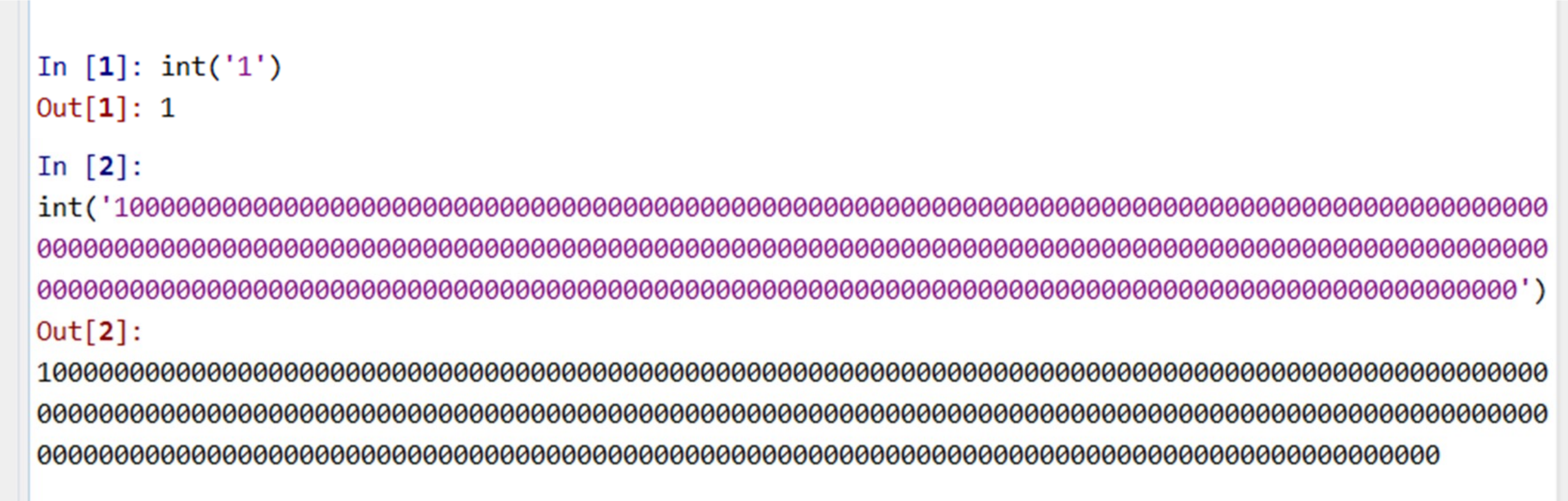
https://www.anaconda.com/download/

Data Types

type	description	mutable ?	indexing?
int	integer number	NO	N.A.
float	real number within a finite range	NO	N.A.
string	a sequence of characters	NO	character-indexing
list	a sequence of objects	YES	element-indexing
tuple	a sequence of objects	NO	element-indexing
range	a sequence of integers	NO	element-indexing
set	a collection of unique objects	YES	N.A.
dictionary	a collection of key : value pairs	YES	N.A.

int

- an **int** number in Python can represent any integer of any length



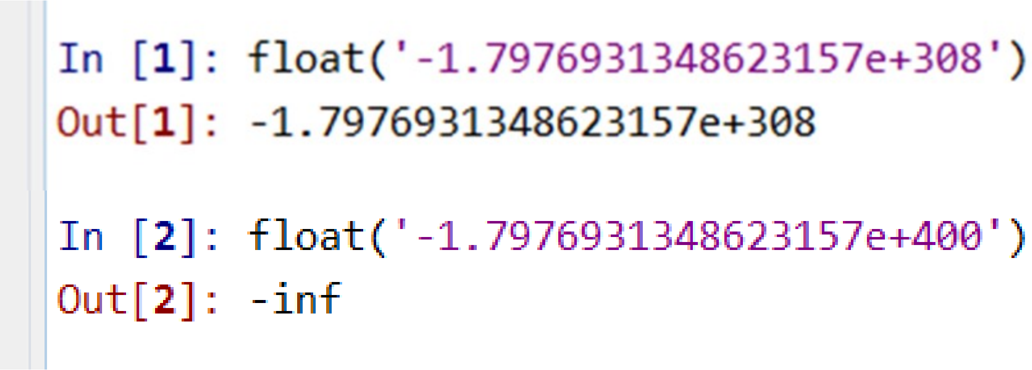
```
In [1]: int('1')
Out[1]: 1

In [2]:
int('1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000')
Out[2]:
100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

int(string) is a function that can convert a string to an integer (int)

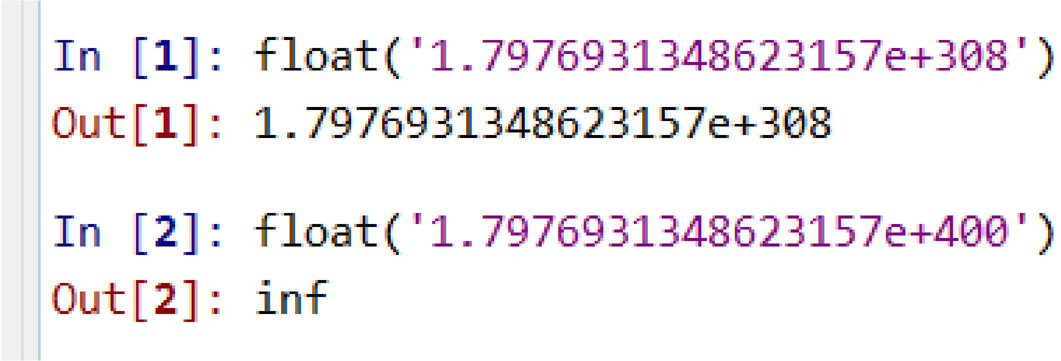
float

- **float** numbers in Python can only represent a subset of real numbers
- In a 64-bit computer, a float number has 64-bits

type	min	max
float	-1.7976931348623157e+308	1.7976931348623157e+308



```
In [1]: float('-1.7976931348623157e+308')
Out[1]: -1.7976931348623157e+308

In [2]: float('-1.7976931348623157e+400')
Out[2]: -inf
```



```
In [1]: float('1.7976931348623157e+308')
Out[1]: 1.7976931348623157e+308

In [2]: float('1.7976931348623157e+400')
Out[2]: inf
```

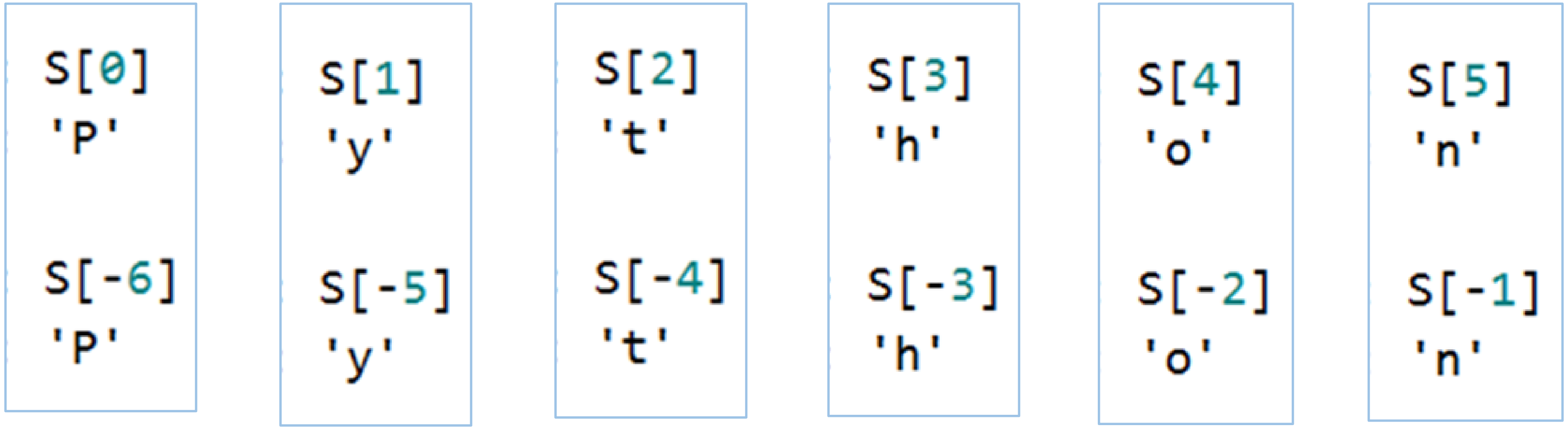
string

- A string is a sequence of **char**acters
- Any characters on the computer keyboard can be put into a string

S = "Python"

Char	P	y	t	h	o	n
Non-negative Index	0	1	2	3	4	5
Negative Index	-6	-5	-4	-3	-2	-1

get a character by index



```
S[0]
'P'

S[-6]
'P'
```

```
S[1]
'y'

S[-5]
'y'
```

```
S[2]
't'

S[-4]
't'
```

```
S[3]
'h'

S[-3]
'h'
```

```
S[4]
'o'

S[-2]
'o'
```

```
S[5]
'n'

S[-1]
'n'
```

string

- Obtain a Sub-string of a String (a.k.a. slicing)

S = "Python Data"

	P	y	t	h	o	n		D	a	t	a
Non-negative Index →	0	1	2	3	4	5	6	7	8	9	10
Negative Index →	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

S[0:6] is "Python"

S[7:11] is "Data"

S[-11:-5] is "Python"

S[-4:-1] is "Dat"

string

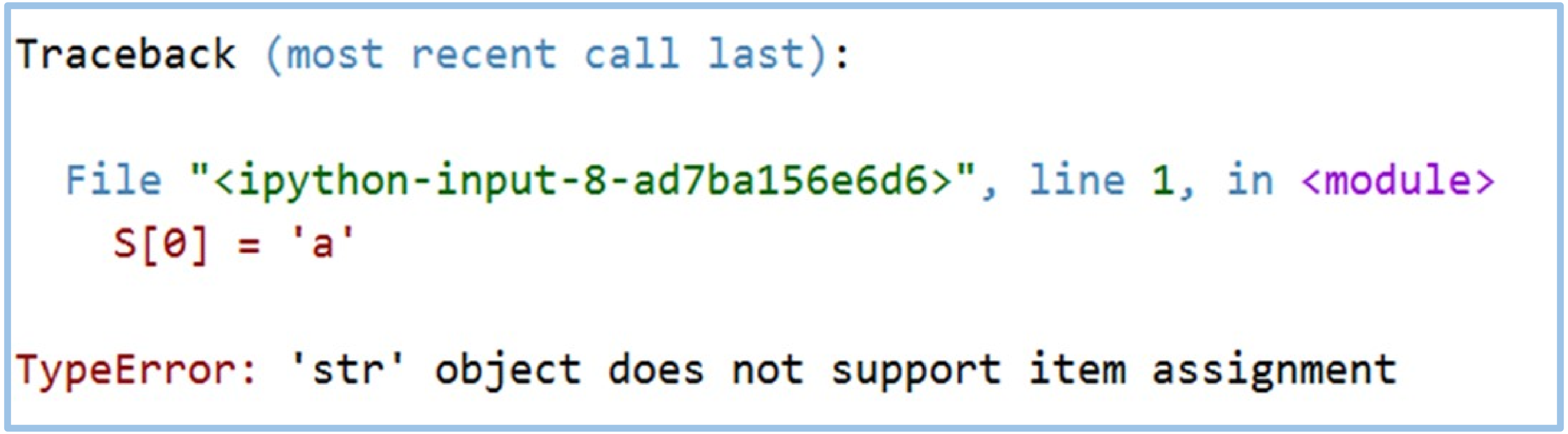
- a string is **immutable**

```
S = "Python"
```

try to modify a character

```
S[0] = "a"
```

Error Message from Python



```
Traceback (most recent call last):

  File "<ipython-input-8-ad7ba156e6d6>", line 1, in <module>
    S[0] = 'a'

TypeError: 'str' object does not support item assignment
```

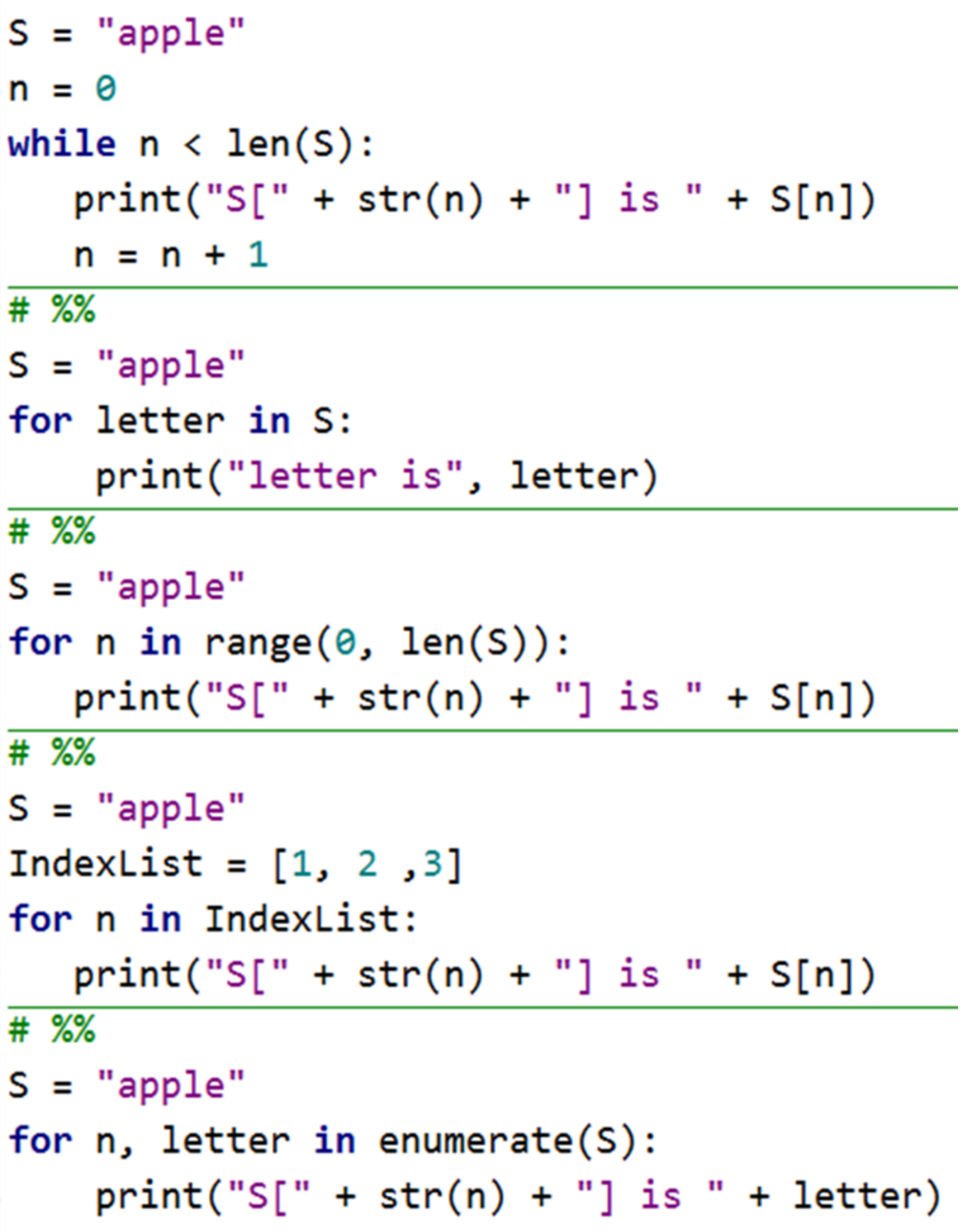
Loop Over a String: many choices …

many choices to loop over a string S…

a **while** loop

4 kinds of **for** loop

use whichever you like/need

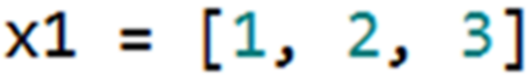


```
S = "apple"
n = 0
while n < len(S):
   print("S[" + str(n) + "] is " + S[n])
   n = n + 1
# %%
S = "apple"
for letter in S:
    print("letter is", letter)
# %%
S = "apple"
for n in range(0, len(S)):
   print("S[" + str(n) + "] is " + S[n])
# %%
S = "apple"
IndexList = [1, 2 ,3]
for n in IndexList:
   print("S[" + str(n) + "] is " + S[n])
# %%
S = "apple"
for n, letter in enumerate(S):
    print("S[" + str(n) + "] is " + letter)
```

list

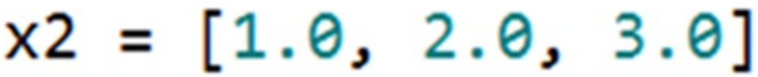
- A list is a sequence of objects/elements
- An element of a list can be any object in Python

(1) int



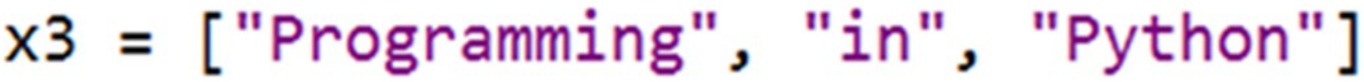
```
x1 = [1, 2, 3]
```

(2) float



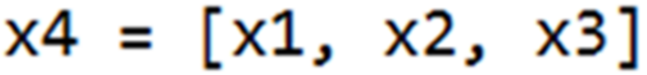
```
x2 = [1.0, 2.0, 3.0]
```

(3) string



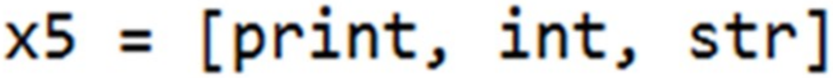
```
x3 = ["Programming", "in", "Python"]
```

(4) list



```
x4 = [x1, x2, x3]
```

(5) function



```
x5 = [print, int, str]
```

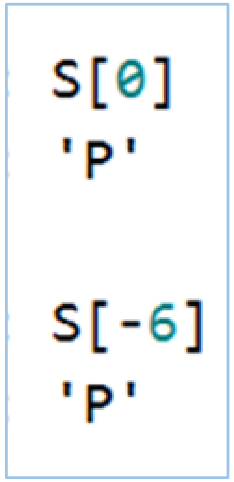
- A list is also called a container

list

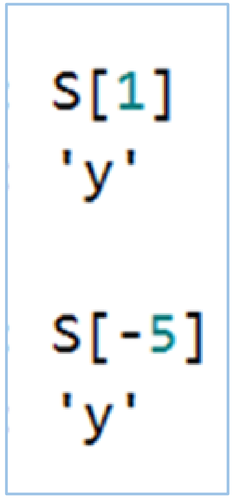
```
S = ['P', 'y', 't', 'h', 'o', 'n']
```

Element	P	y	t	h	o	n
Non-negative Index	**0**	**1**	**2**	**3**	**4**	**5**
Negative Index	**-6**	**-5**	**-4**	**-3**	**-2**	**-1**

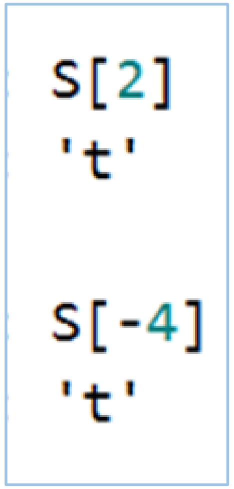
get an element by index



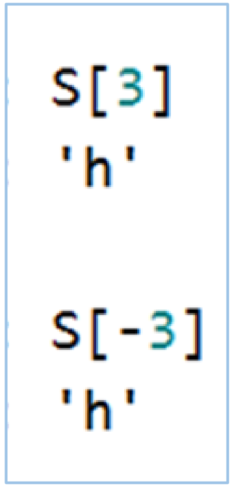
```
S[0]
'P'

S[-6]
'P'
```



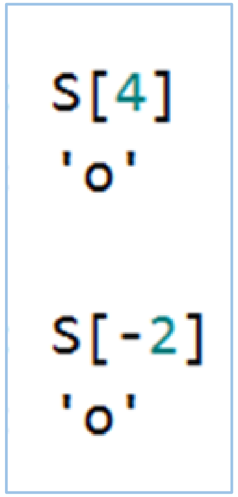
```
S[1]
'y'

S[-5]
'y'
```



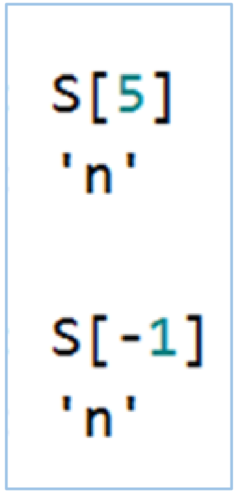
```
S[2]
't'

S[-4]
't'
```



```
S[3]
'h'

S[-3]
'h'
```



```
S[4]
'o'

S[-2]
'o'
```



```
S[5]
'n'

S[-1]
'n'
```

list

- Obtain a Sub-list of a List (a.k.a. slicing)

```
S = ['P', 'y', 't', 'h', 'o', 'n', ' ' , 'D', 'a', 't', 'a']
```

	P	y	t	h	o	n		D	a	t	a
Non-negative Index	0	1	2	3	4	5	6	7	8	9	10
Negative Index	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
S[0:6] is ['P', 'y', 't', 'h', 'o', 'n']
```

```
S[7:11] is ['D', 'a', 't', 'a']
```

```
S[-11:-5] is ['P', 'y', 't', 'h', 'o', 'n']
```

```
S[-4:-1] is ['D', 'a', 't']
```

list

- a list is **mutable**

```
S = [0, 1, 2, 3, 4, 5]
```

modify an **element**

```
S[0] = 10
```

now, the list is

```
S is [10, 1, 2, 3, 4, 5]
```

```
S = [0, 1, 2, 3, 4, 5]
```

modify a **sub-list**

```
S[0:2] = [10, 11]
```

now, the list is

```
S is [10, 11, 2, 3, 4, 5]
```

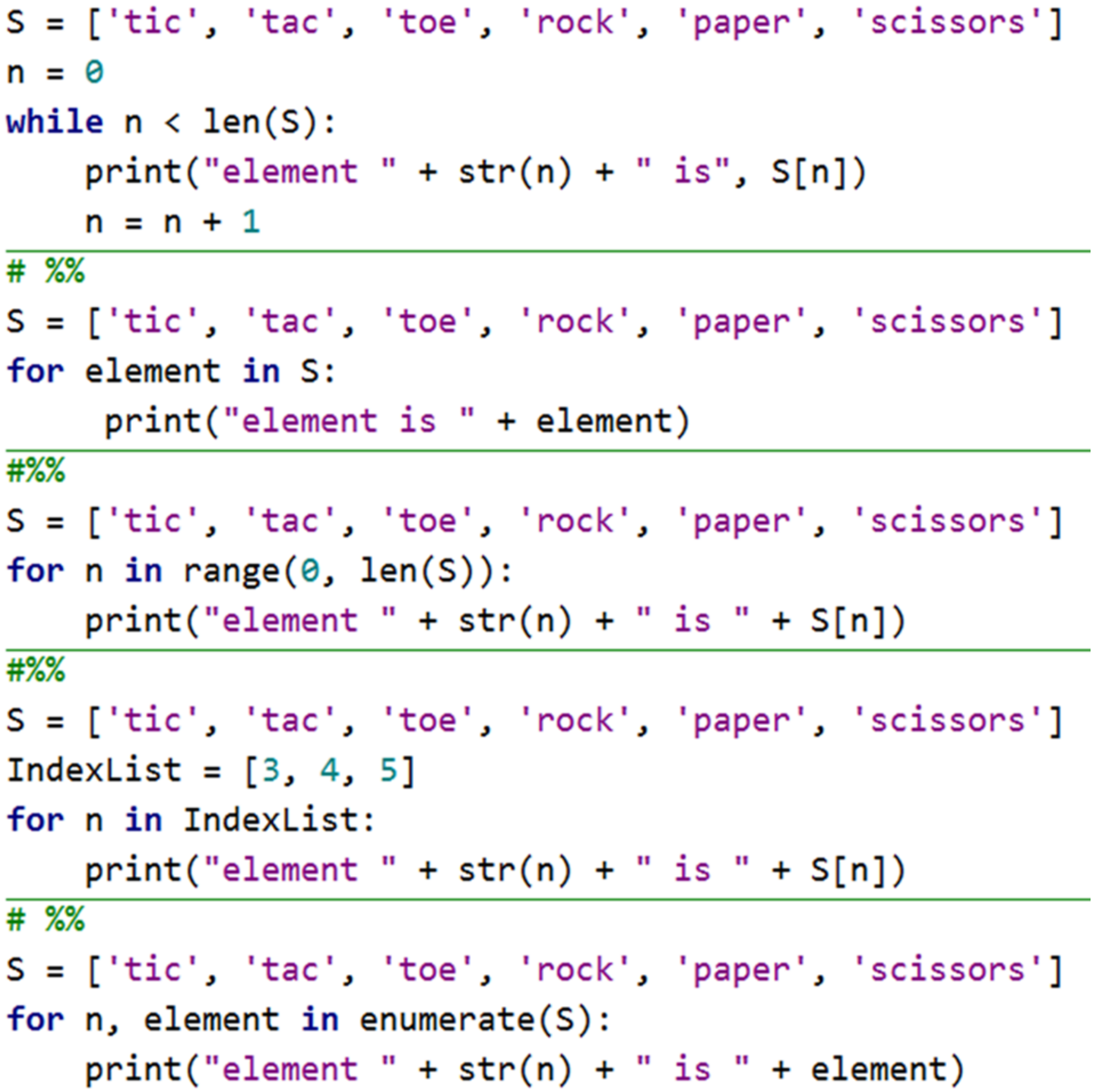
Loop Over a List: many choices

many choices to loop over a list S

a **while** loop

4 kinds of **for** loop

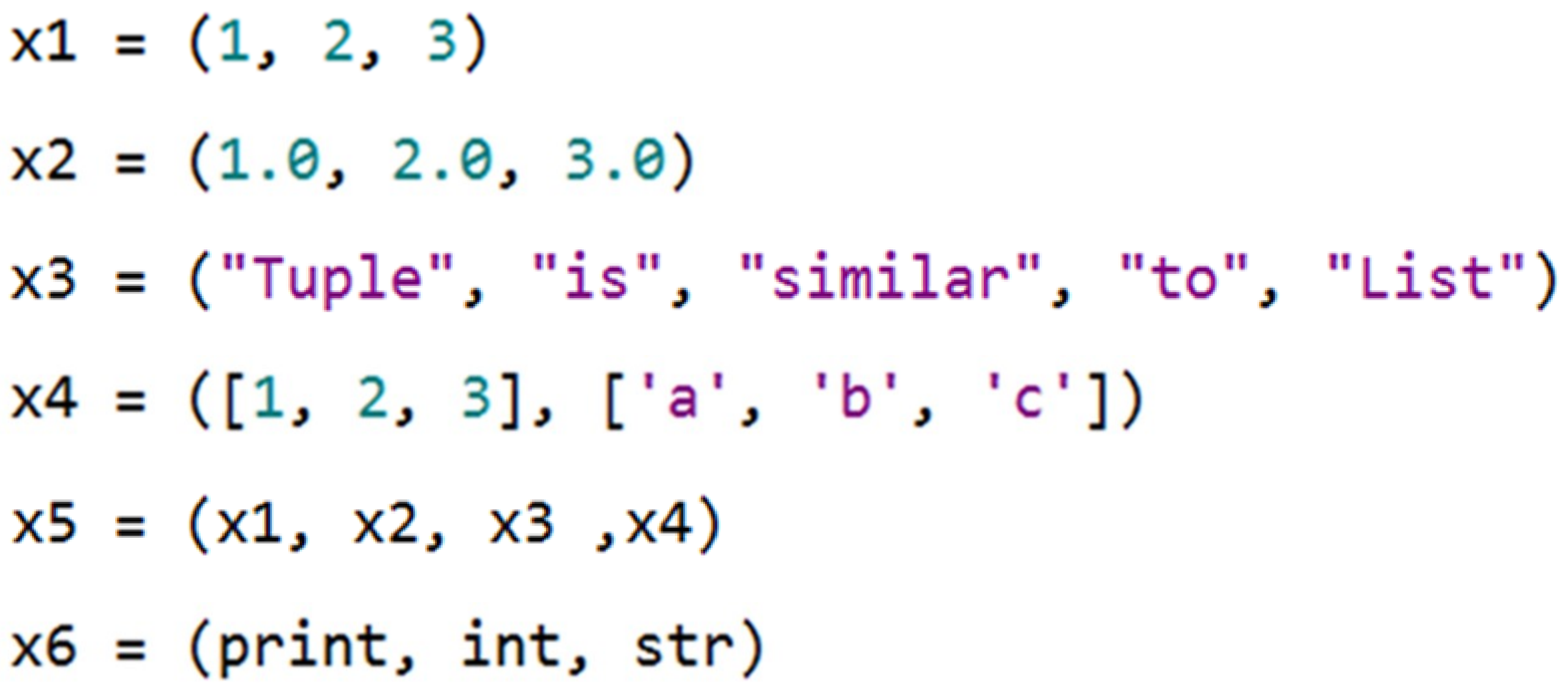
use whichever you like/need



```
S = ['tic', 'tac', 'toe', 'rock', 'paper', 'scissors']
n = 0
while n < len(S):
    print("element " + str(n) + " is", S[n])
    n = n + 1
# %%
S = ['tic', 'tac', 'toe', 'rock', 'paper', 'scissors']
for element in S:
     print("element is " + element)
#%%
S = ['tic', 'tac', 'toe', 'rock', 'paper', 'scissors']
for n in range(0, len(S)):
    print("element " + str(n) + " is " + S[n])
#%%
S = ['tic', 'tac', 'toe', 'rock', 'paper', 'scissors']
IndexList = [3, 4, 5]
for n in IndexList:
    print("element " + str(n) + " is " + S[n])
# %%
S = ['tic', 'tac', 'toe', 'rock', 'paper', 'scissors']
for n, element in enumerate(S):
    print("element " + str(n) + " is " + element)
```

tuple

- A tuple is a sequence of objects (similar to a list)
- An element of a tuple can be any object in Python
- A tuple is **immutable**



(1) int `x1 = (1, 2, 3)`

(2) float `x2 = (1.0, 2.0, 3.0)`

(3) string `x3 = ("Tuple", "is", "similar", "to", "List")`

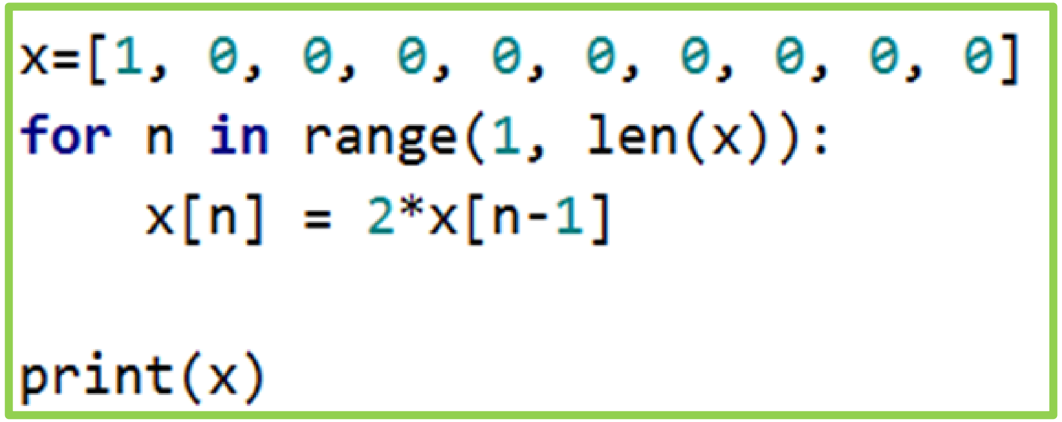
(4) list `x4 = ([1, 2, 3], ['a', 'b', 'c'])`

(5) tuple `x5 = (x1, x2, x3 ,x4)`

(6) function `x6 = (print, int, str)`

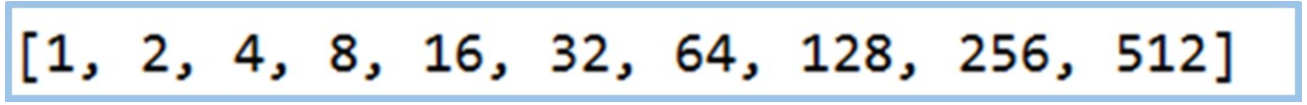
range

- a range object is created by the function range(a, b, c)
- a range object is often used in a for loop



```
x=[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
for n in range(1, len(x)):
    x[n] = 2*x[n-1]

print(x)
```

Output



```
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
```

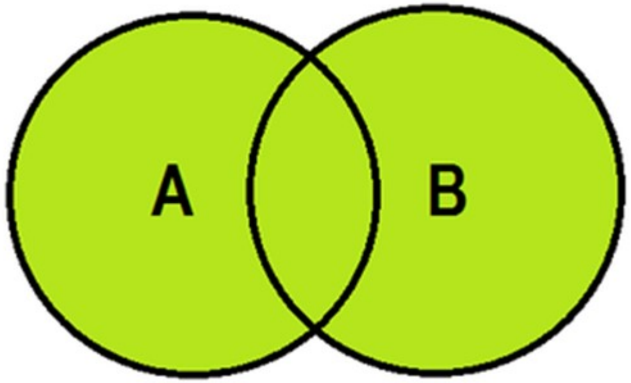
set

- A set is an unordered collection of distinct objects
- The objects of a set are called elements

Example: Set1 = {3, 5, 6, 7, 1} = {1, 3, 5, 6, 7}

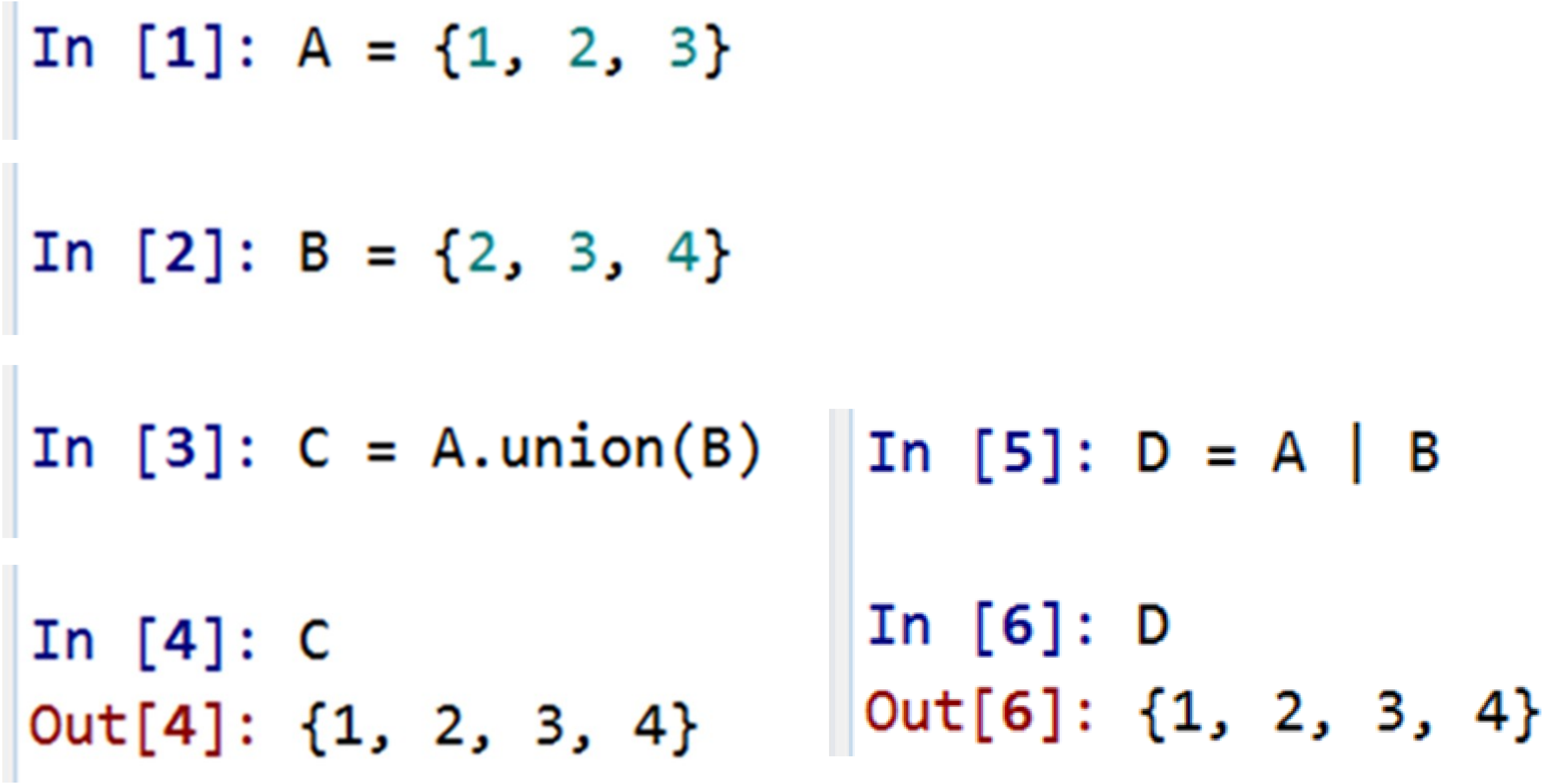
Set Operation: **union**

www.hackerrank.com



A.union(B) or A|B

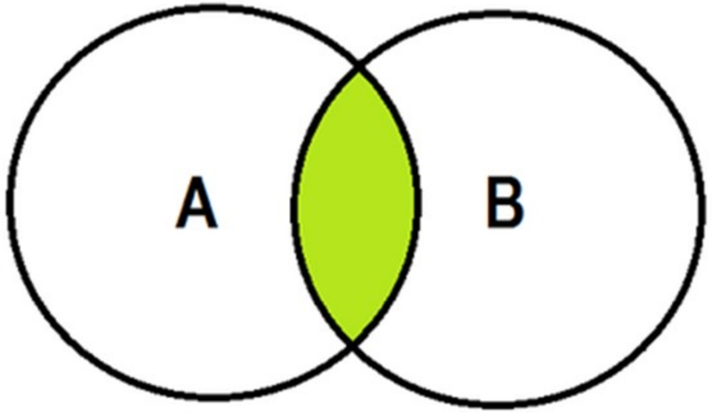
x in the set C=A|B
x **in** A **or** x **in** B



```
In [1]: A = {1, 2, 3}

In [2]: B = {2, 3, 4}

In [3]: C = A.union(B)

In [4]: C
Out[4]: {1, 2, 3, 4}
```

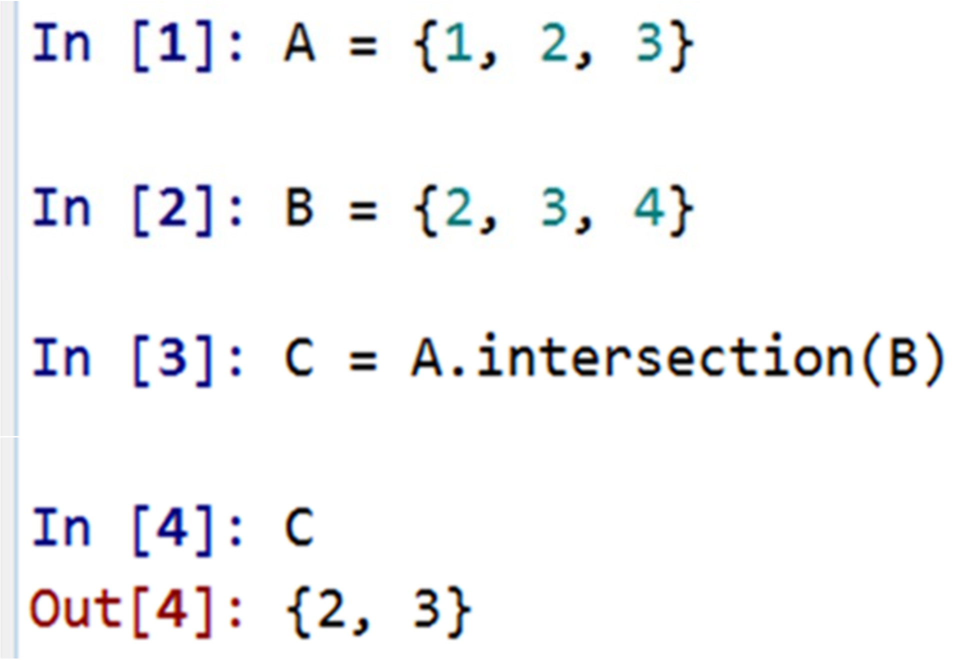
```
In [5]: D = A | B

In [6]: D
Out[6]: {1, 2, 3, 4}
```

Set Operation: **intersection**

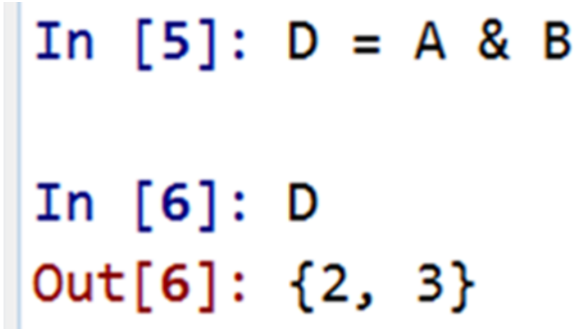


A.intersection(B) or A&B

x in the set C=A&B
x **in** A **and** x **in** B

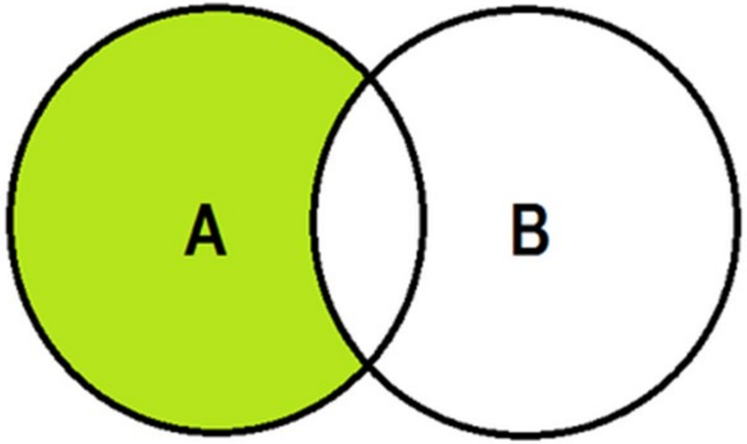


```
In [1]: A = {1, 2, 3}

In [2]: B = {2, 3, 4}

In [3]: C = A.intersection(B)

In [4]: C
Out[4]: {2, 3}
```



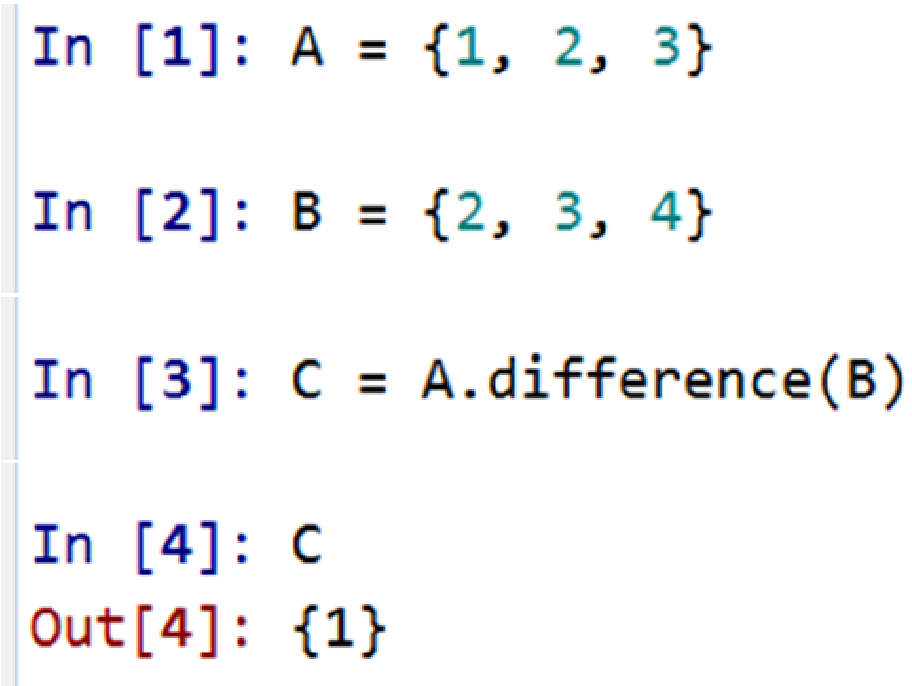
```
In [5]: D = A & B

In [6]: D
Out[6]: {2, 3}
```

Set Operation: **difference**

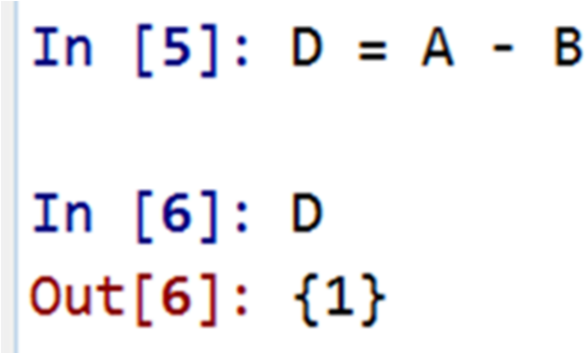


A.difference(B) or A - B

x in the set C=A-B
x **in** A **and** x **not in** B

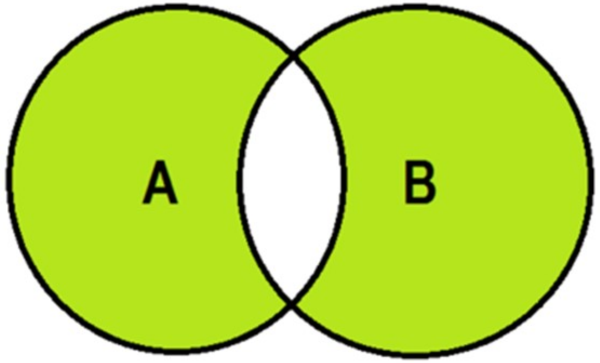


```
In [1]: A = {1, 2, 3}

In [2]: B = {2, 3, 4}

In [3]: C = A.difference(B)

In [4]: C
Out[4]: {1}
```



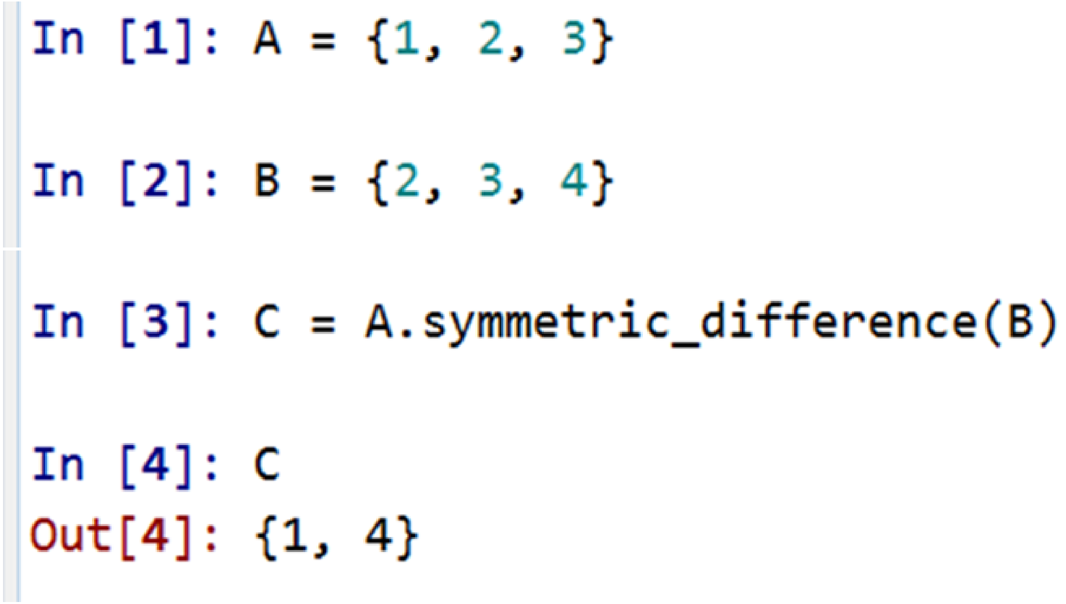
```
In [5]: D = A - B

In [6]: D
Out[6]: {1}
```

Set Operation: **symmetric difference**

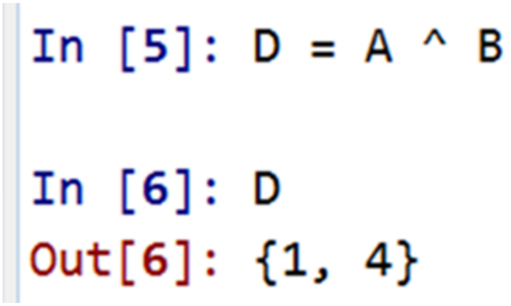


A.symmetric_difference(B) or A^B

x in the set C=A^B
(x **in** A **and** x **not in** B)
or (x **in** B **and** x **not in** A)



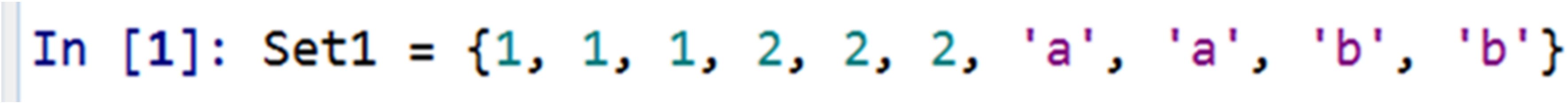
```
In [1]: A = {1, 2, 3}

In [2]: B = {2, 3, 4}

In [3]: C = A.symmetric_difference(B)

In [4]: C
Out[4]: {1, 4}
```



```
In [5]: D = A ^ B

In [6]: D
Out[6]: {1, 4}
```

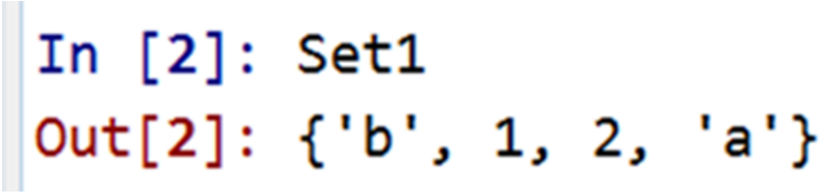
A set has no duplicate elements

define a set with duplicate numbers and strings



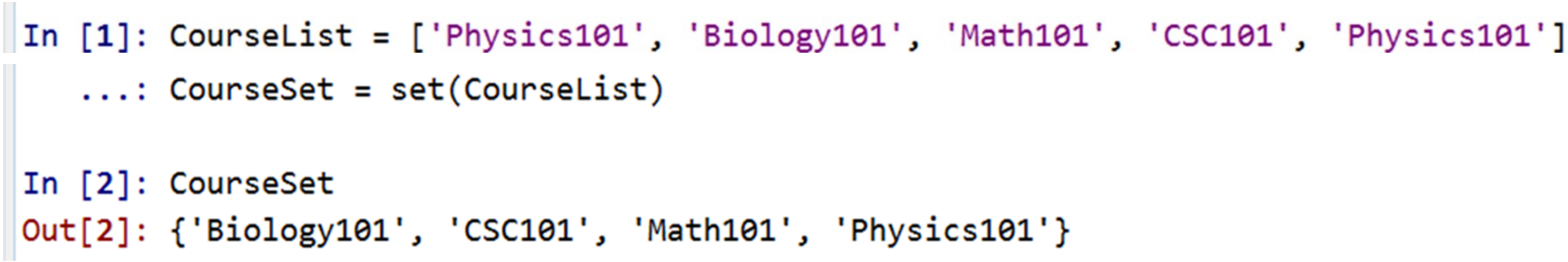
```
In [1]: Set1 = {1, 1, 1, 2, 2, 2, 'a', 'a', 'b', 'b'}
```

Python removes redundant copies of the elements

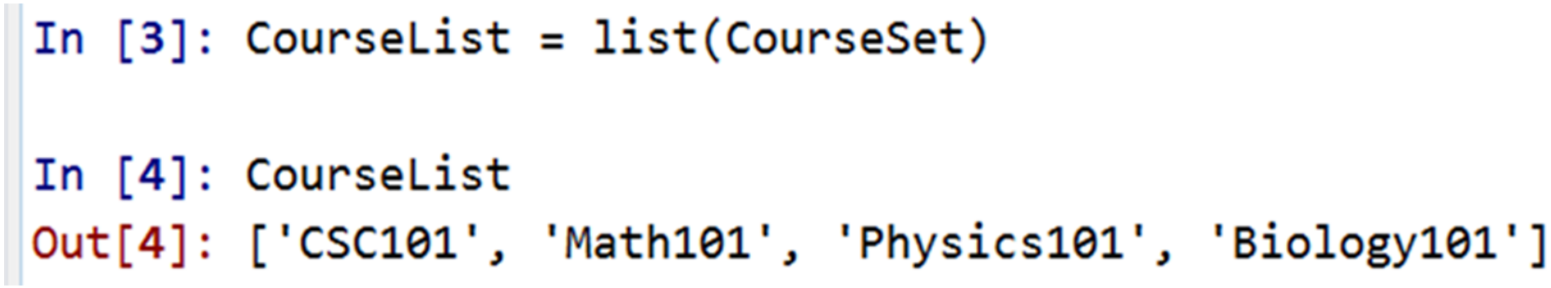


```
In [2]: Set1
Out[2]: {'b', 1, 2, 'a'}
```

Use a set to remove duplicate elements in a list



```
In [1]: CourseList = ['Physics101', 'Biology101', 'Math101', 'CSC101', 'Physics101']
   ...: CourseSet = set(CourseList)

In [2]: CourseSet
Out[2]: {'Biology101', 'CSC101', 'Math101', 'Physics101'}
```

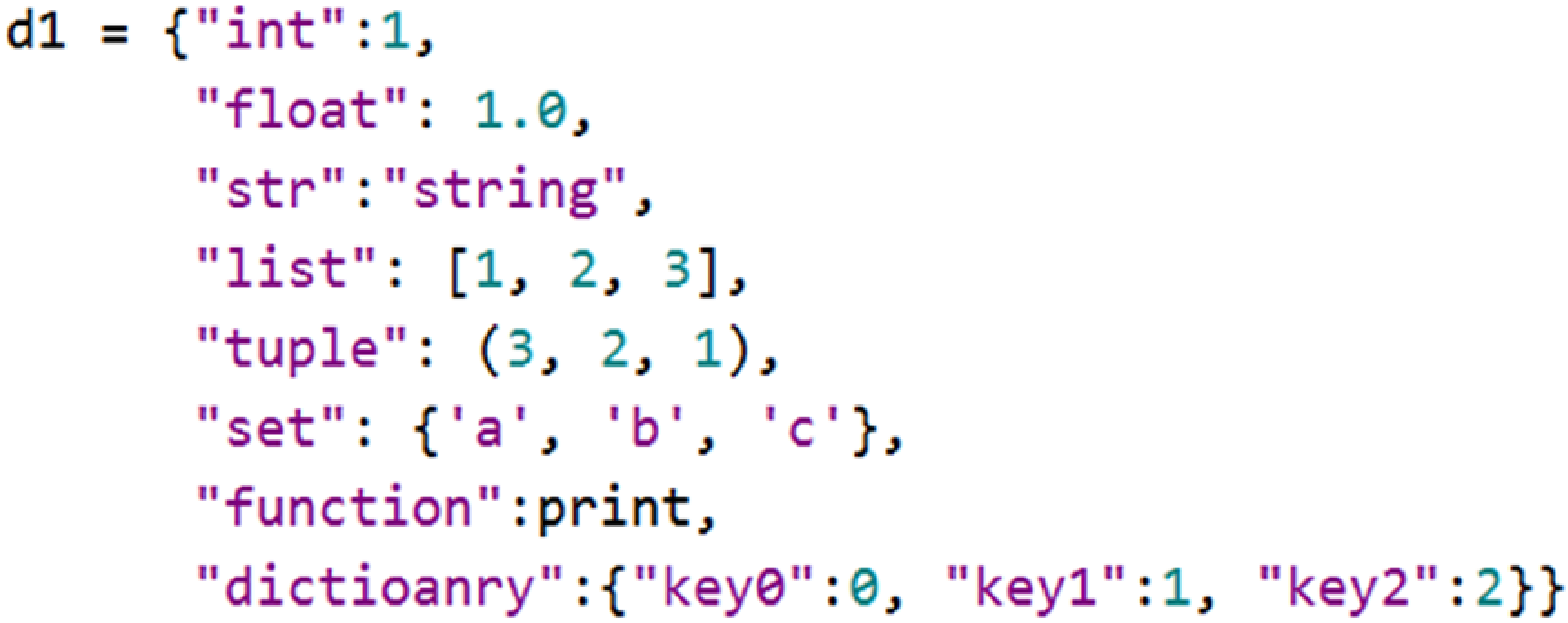


```
In [3]: CourseList = list(CourseSet)

In [4]: CourseList
Out[4]: ['CSC101', 'Math101', 'Physics101', 'Biology101']
```

dictionary

- A dictionary is a collection of ***key: value*** pairs
  x={"name": "Python", "version": 3.7}

- A **value** in a dictionary can be any object in Python

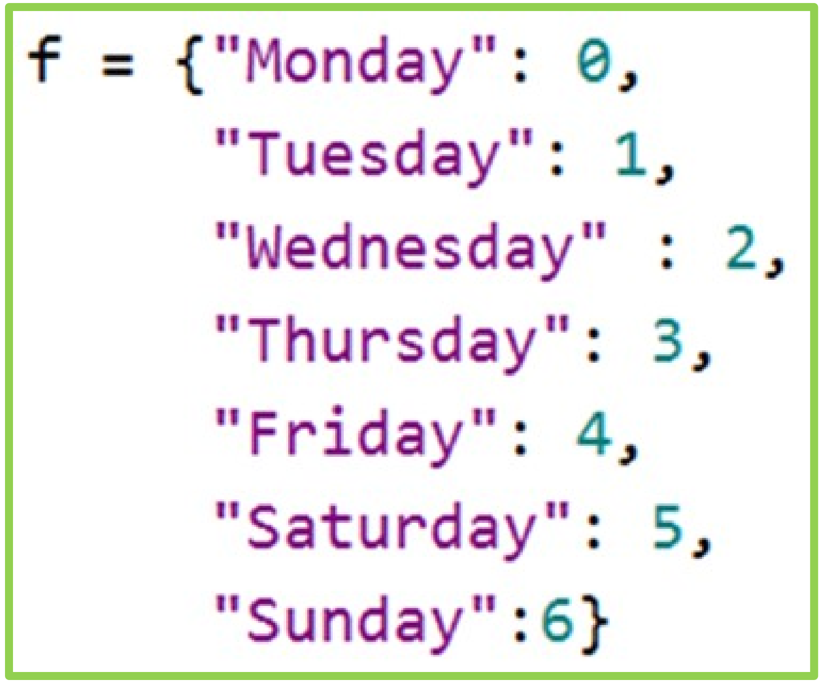


```
d1 = {"int":1,
      "float": 1.0,
      "str":"string",
      "list": [1, 2, 3],
      "tuple": (3, 2, 1),
      "set": {'a', 'b', 'c'},
      "function":print,
      "dictioanry":{"key0":0, "key1":1, "key2":2}}
```

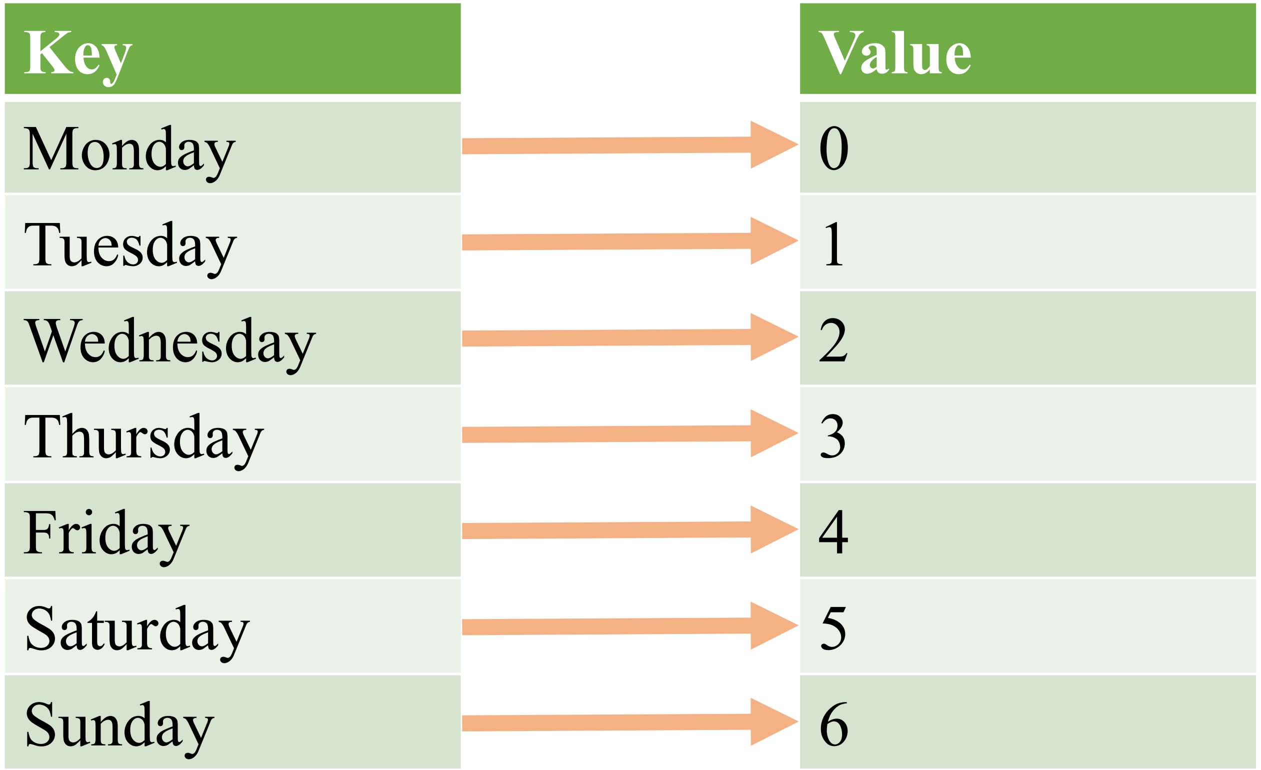
a dictionary is a mapping from Key to Value

- (Math) a mapping from x to y is a function: y = f(x)

map the name of a day to a number



```
f = {"Monday": 0,
     "Tuesday": 1,
     "Wednesday" : 2,
     "Thursday": 3,
     "Friday": 4,
     "Saturday": 5,
     "Sunday":6}
```



Key	Value
Monday	0
Tuesday	1
Wednesday	2
Thursday	3
Friday	4
Saturday	5
Sunday	6

f is the name of the dictionary object

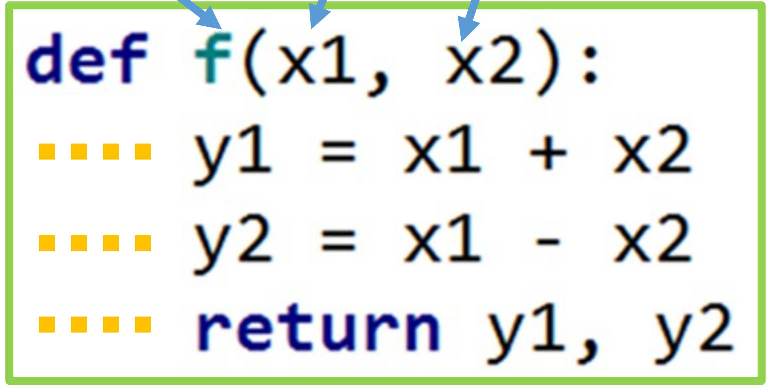
- Access a **Value** by **Key**: **f["Monday"]** is **0**

Arguments, Parameters and Returns of a Function

function name Parameters

use **def** to define a function

4 blank spaces for indentation

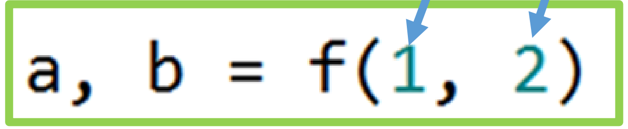


```
def f(x1, x2):
····y1 = x1 + x2
····y2 = x1 - x2
····return y1, y2
```

multiple returns

Arguments

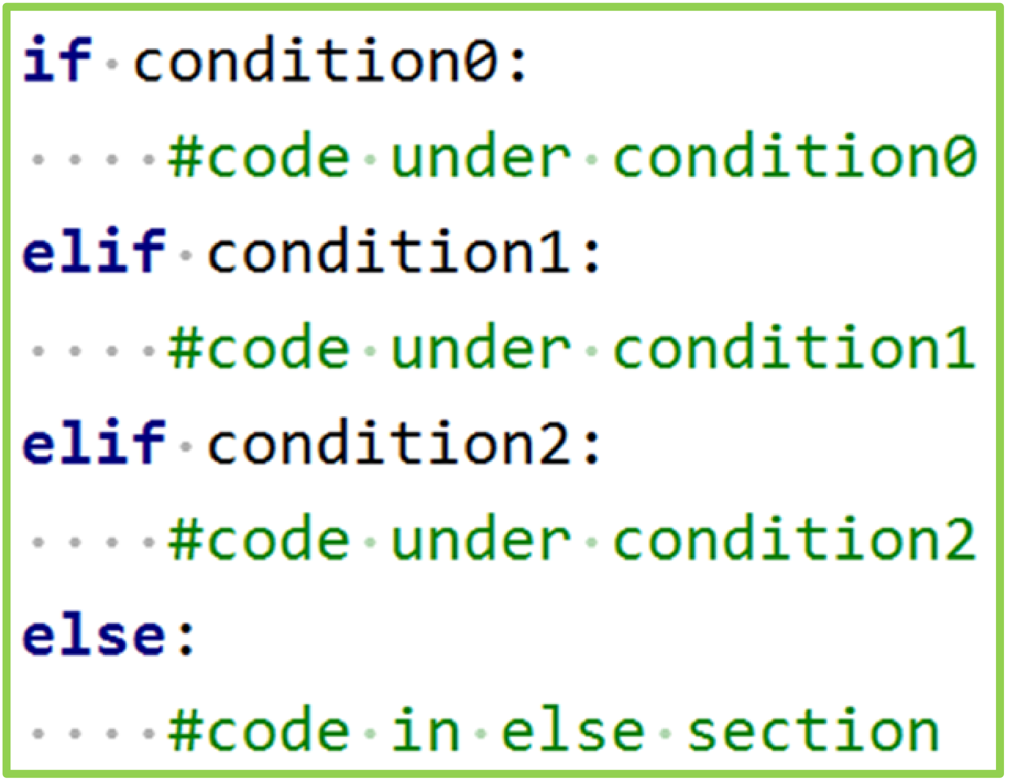
call/**run**/**use the function** by its name and pass arguments



```
a, b = f(1, 2)
```

Conditional Execution Using an **if**, **elif**, **else** Block

an **if**, **elif**, **else** Block



```
if condition0:
    #code under condition0
elif condition1:
    #code under condition1
elif condition2:
    #code under condition2
else:
    #code in else section
```

- Python will check each boolean condition (**if** / **elif**) from top to bottom
- If a condition is **True**, then the code under that condition is executed, and other **elif** / **else** sections are ignored
- If every condition is **False**, then the code in the **else** section will run

online reference

https://www.python-course.eu/python3_course.php

https://jakevdp.github.io/PythonDataScienceHandbook/