Introduction to Python

章宇 ZHANG Yu
y.zhang@swufe.edu.cn

Installation

- Anaconda
  - Python, Jupyter, Spyder

- Gurobi

Anaconda (for Python)

- It includes over 330 Python and R packages

- It includes
  - Integrated Development Environment (Spyder)
  - The leading web interactive notebook for data science (Jupyter).

Gurobi

- A commercial optimization solver for
  - linear programming (LP),
  - quadratic programming (QP),
  - quadratically constrained programming (QCP),
  - mixed-integer linear programming (MILP),
  - mixed-integer quadratic programming (MIQP),
  - mixed-integer quadratically constrained programming (MIQCP).

LOP Example

- A furniture company makes products
- Production require wood, finishing labor and carpentry labor.

	Desk	Table	Chair	Avail.
Profit	60	30	20	
Wood	8	6	1	48
Finish Hrs	4	2	1.5	20
Carpentry Hrs	2	1.5	0.5	8

LOP Example

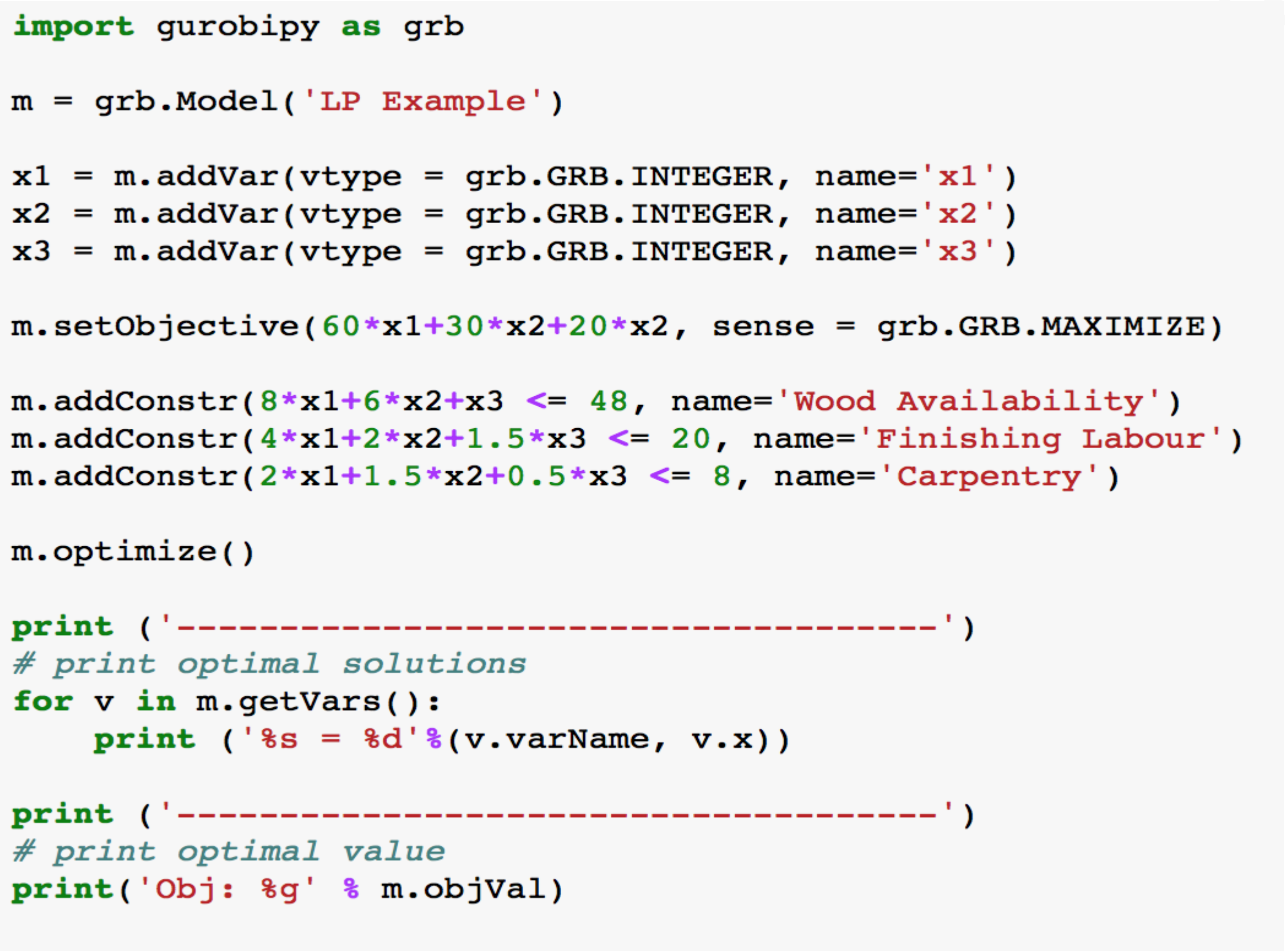
Decision variables:

$x_1 =$ Num. desks, $x_2 =$ Num. tables

$x_3 =$ Num. chairs

$$\begin{aligned} \max \quad & 60x_1 + 30x_2 + 20x_3 & \\ \text{s.t.} \quad & 8x_1 + 6x_2 + x_3 & \leq 48 \\ & 4x_1 + 2x_2 + 1.5x_3 & \leq 20 \\ & 2x_1 + 1.5x_2 + 0.5x_3 & \leq 8 \\ & x_1, x_2, x_3 & \geq 0, \end{aligned}$$

Python + Gurobi API



```
import gurobipy as grb

m = grb.Model('LP Example')

x1 = m.addVar(vtype = grb.GRB.INTEGER, name='x1')
x2 = m.addVar(vtype = grb.GRB.INTEGER, name='x2')
x3 = m.addVar(vtype = grb.GRB.INTEGER, name='x3')

m.setObjective(60*x1+30*x2+20*x2, sense = grb.GRB.MAXIMIZE)

m.addConstr(8*x1+6*x2+x3 <= 48, name='Wood Availability')
m.addConstr(4*x1+2*x2+1.5*x3 <= 20, name='Finishing Labour')
m.addConstr(2*x1+1.5*x2+0.5*x3 <= 8, name='Carpentry')

m.optimize()

print ('------------------------------------')
# print optimal solutions
for v in m.getVars():
    print ('%s = %d'%(v.varName, v.x))

print ('------------------------------------')
# print optimal value
print('Obj: %g' % m.objVal)
```

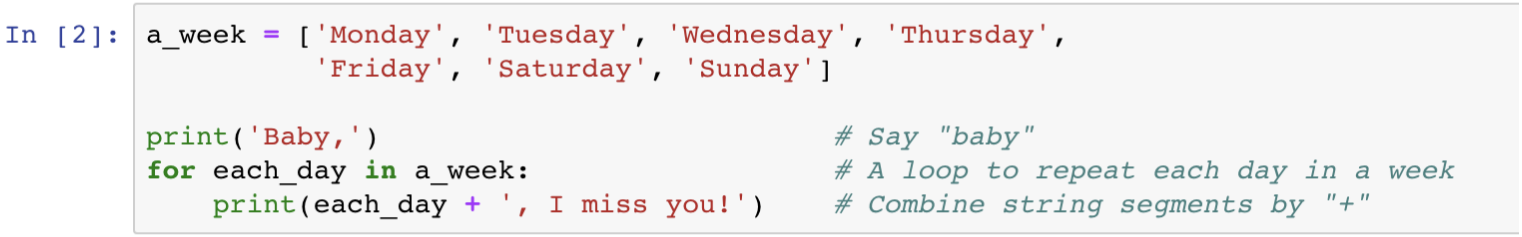
Python Basics

- Python is an interpreted high-level programming language for general-purpose programming.

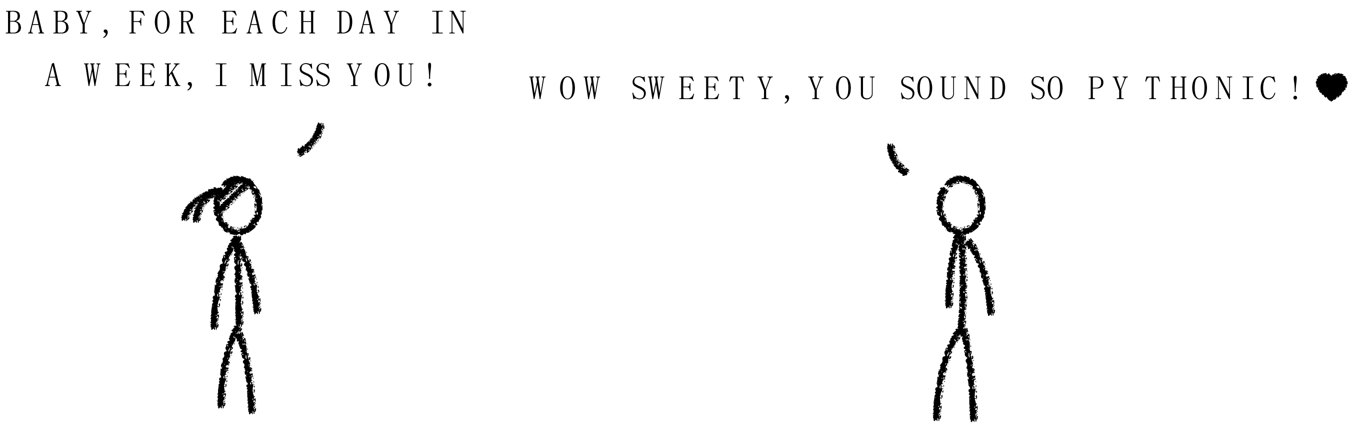
- Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Good Code Readability

- Being “Pythonic”

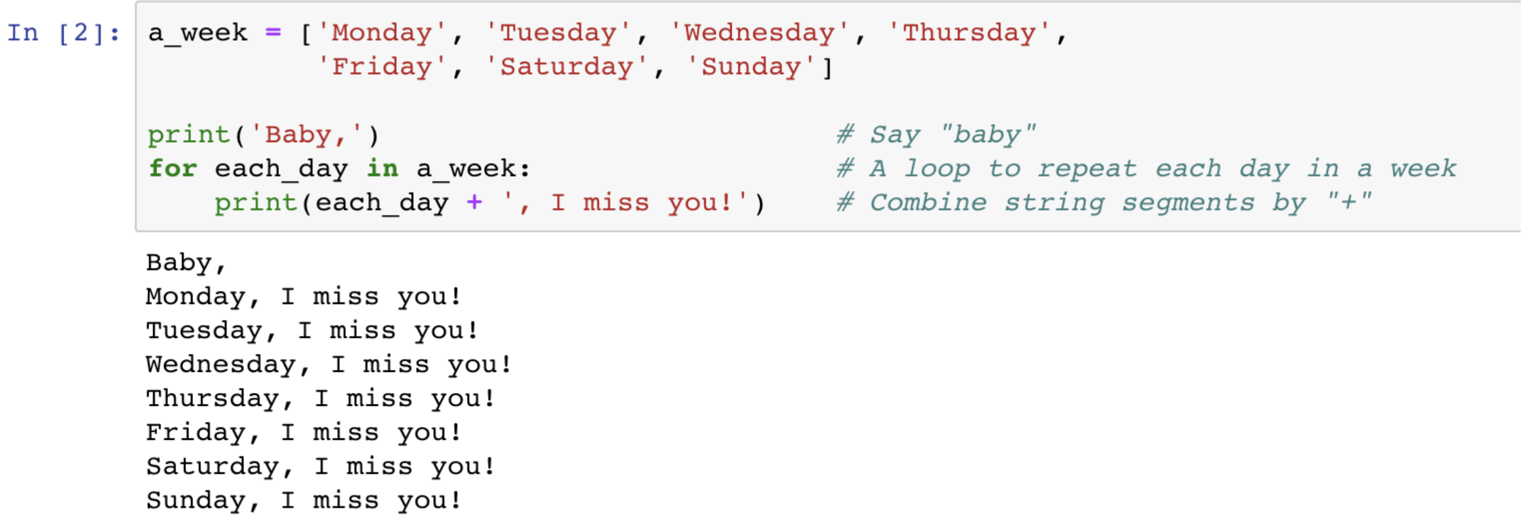


```
In [2]: a_week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
                  'Friday', 'Saturday', 'Sunday']

        print('Baby,')                          # Say "baby"
        for each_day in a_week:                 # A loop to repeat each day in a week
            print(each_day + ', I miss you!')   # Combine string segments by "+"
```


BABY, FOR EACH DAY IN A WEEK, I MISS YOU!
WOW SWEETY, YOU SOUND SO PYTHONIC! ♥

Good Code Readability

- Being “Pythonic”



```
In [2]: a_week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
                  'Friday', 'Saturday', 'Sunday']

        print('Baby,')                              # Say "baby"
        for each_day in a_week:                     # A loop to repeat each day in a week
            print(each_day + ', I miss you!')       # Combine string segments by "+"
```

```
Baby,
Monday, I miss you!
Tuesday, I miss you!
Wednesday, I miss you!
Thursday, I miss you!
Friday, I miss you!
Saturday, I miss you!
Sunday, I miss you!
```

  - Comments
  - Meaningful variable names

Python Basics

- …
- Python has a design philosophy that emphasizes code readability, notably using significant whitespace.
- It has a large and comprehensive standard library.

Library

- Ancient programming languages

GUYS, I NEED A WHEEL FOR MY BICYCLE!
I HAVE A SCREWDRIVER!
I HAVE SCREWS.
IT'S SIMPLE! FRIST, GATHER EVERYTHING: HUB, SPOKES, RIM, SCREWDRIVER, SPOKE WRENCH, RULER AND A BIKE FRAME. THEN YOU GRAB THE HUB …… …… …… …… …… …… …… …… …… ……
THAT'S THE LAST STEP. IT ONLY TAKES ABOUT SIXTY-NINE HOURS, AND YOU HAVE A BRAND NEW BICYCLE WHEEL!

Library

- Python

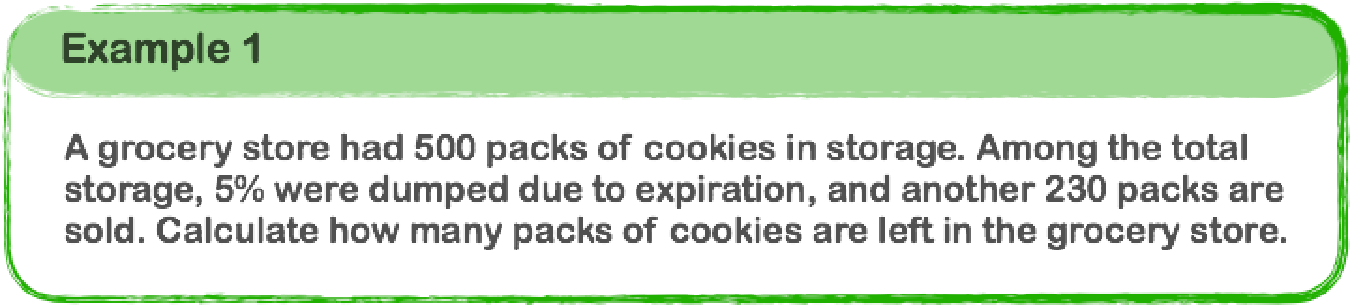
Variables and Basic Types

Python Basic Types

- Numbers
  - int (integers), e.g., -10
  - float (floating point real values), e.g., -10.0, 32.3+e18
- Booleans: True or False
- Strings
  - A contiguous set of characters represented in the quotation marks. e.g., “Hello World!”

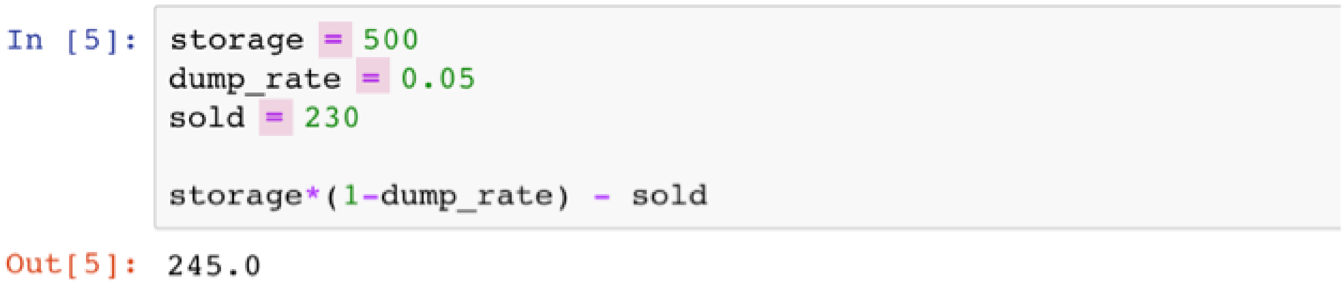
Python Variables

- Variables and Assignment Statements
  - Assignment operator “="



Example 1

A grocery store had 500 packs of cookies in storage. Among the total storage, 5% were dumped due to expiration, and another 230 packs are sold. Calculate how many packs of cookies are left in the grocery store.



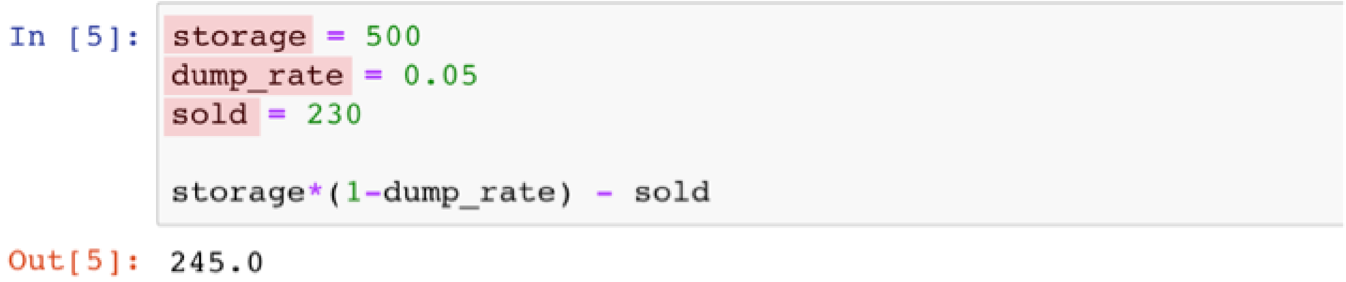
```
In [5]: storage = 500
        dump_rate = 0.05
        sold = 230

        storage*(1-dump_rate) - sold

Out[5]: 245.0
```

Python Variables

- Variables and Assignment Statements
  - Variable(s): name(s) on the left

Example 1

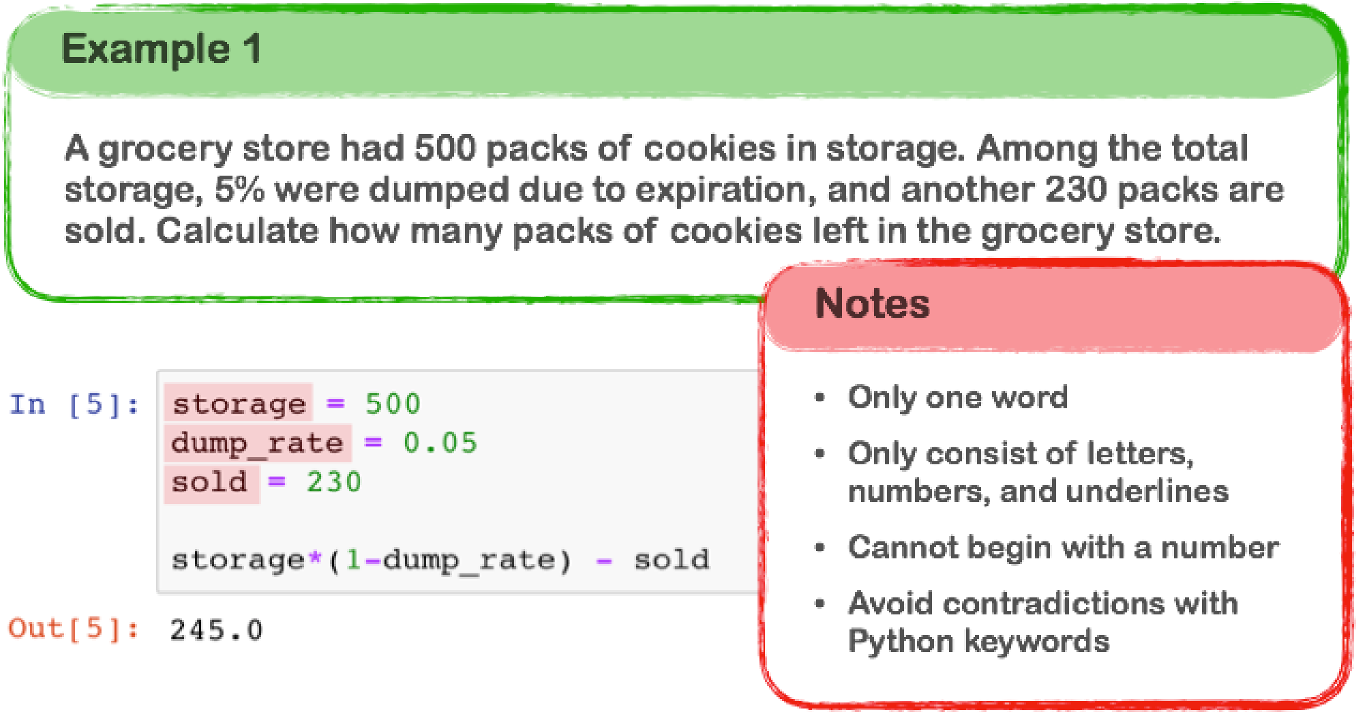
A grocery store had 500 packs of cookies in storage. Among the total storage, 5% were dumped due to expiration, and another 230 packs are sold. Calculate how many packs of cookies are left in the grocery store.



```
In [5]: storage = 500
        dump_rate = 0.05
        sold = 230

        storage*(1-dump_rate) - sold

Out[5]: 245.0
```

Python Variables

- Variables and Assignment Statements
  - Variable(s): name(s) on the left



Example 1

A grocery store had 500 packs of cookies in storage. Among the total storage, 5% were dumped due to expiration, and another 230 packs are sold. Calculate how many packs of cookies left in the grocery store.

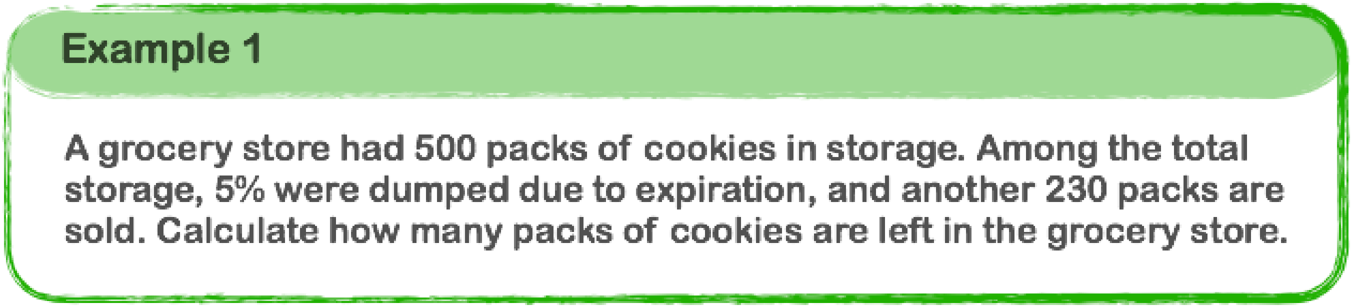
```
In [5]: storage = 500
        dump_rate = 0.05
        sold = 230

        storage*(1-dump_rate) - sold

Out[5]: 245.0
```

Notes

- Only one word
- Only consist of letters, numbers, and underlines
- Cannot begin with a number
- Avoid contradictions with Python keywords

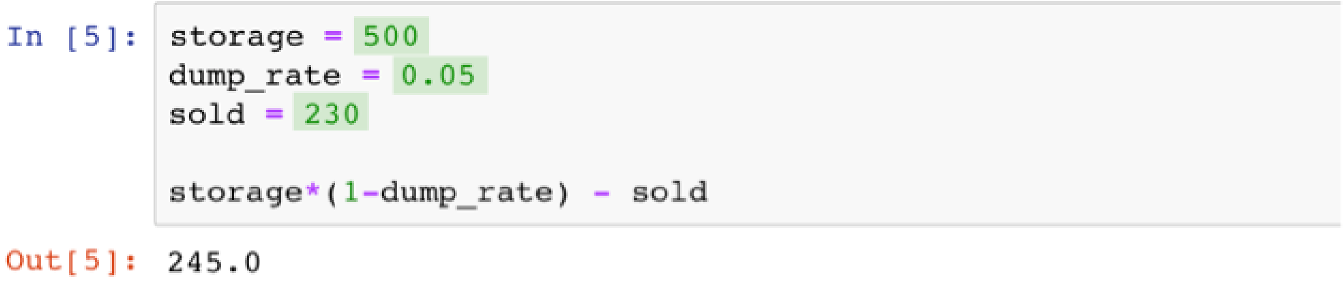
Python Variables

- Variables and Assignment Statements
  - Value(s): expression(s) on the right



Example 1

A grocery store had 500 packs of cookies in storage. Among the total storage, 5% were dumped due to expiration, and another 230 packs are sold. Calculate how many packs of cookies are left in the grocery store.



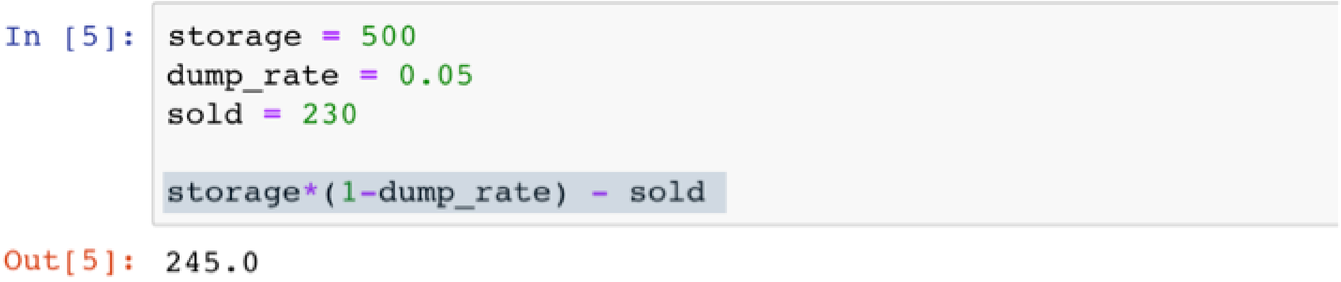
```
In [5]:  storage = 500
          dump_rate = 0.05
          sold = 230

          storage*(1-dump_rate) - sold

Out[5]:  245.0
```

Python Variables

- Variables and Assignment Statements
  - The value of a variable can be retrieved by invoking the name

Example 1

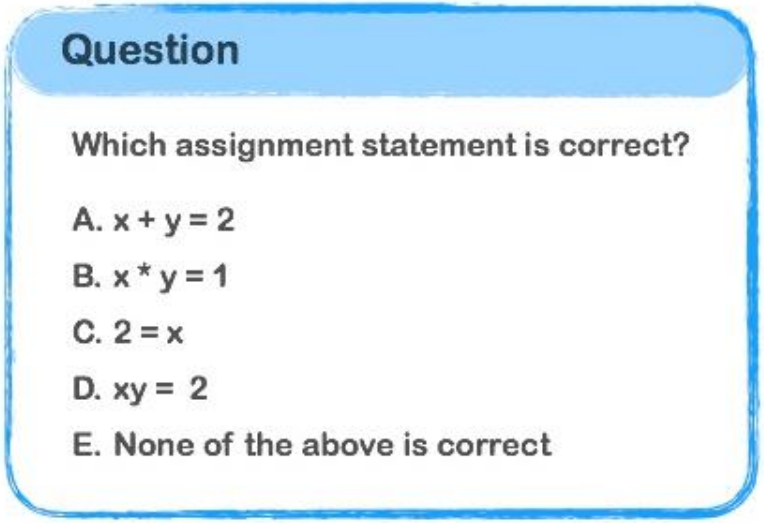
A grocery store had 500 packs of cookies in storage. Among the total storage, 5% were dumped due to expiration, and another 230 packs are sold. Calculate how many packs of cookies are left in the grocery store.



```
In [5]:  storage = 500
         dump_rate = 0.05
         sold = 230

         storage*(1-dump_rate) - sold

Out[5]:  245.0
```

Python Variables

- Variables and Assignment Statements
  - Python vs Math



Question

Which assignment statement is correct?

A. x + y = 2

B. x * y = 1

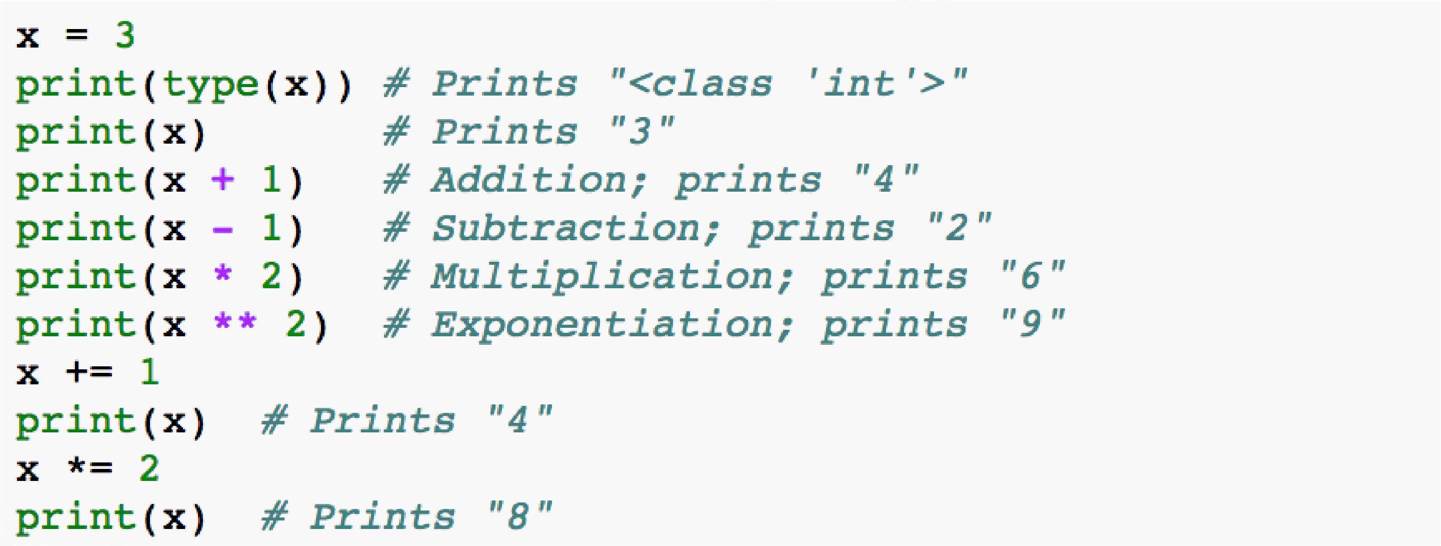
C. 2 = x

D. xy = 2

E. None of the above is correct

Basic Arithmetic

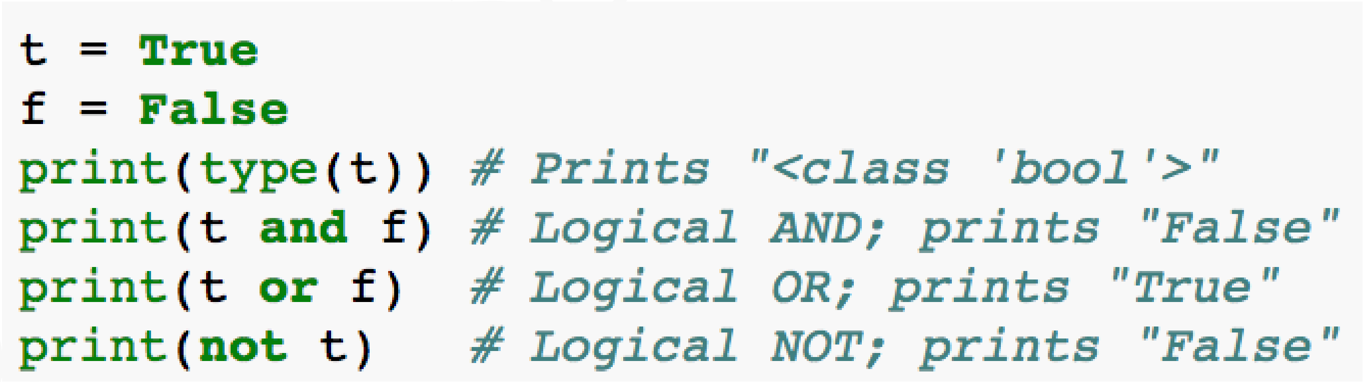
- More examples on basic arithmetic operations



```
x = 3
print(type(x)) # Prints "<class 'int'>"
print(x)       # Prints "3"
print(x + 1)   # Addition; prints "4"
print(x - 1)   # Subtraction; prints "2"
print(x * 2)   # Multiplication; prints "6"
print(x ** 2)  # Exponentiation; prints "9"
x += 1
print(x)  # Prints "4"
x *= 2
print(x)  # Prints "8"
```

Boolean Variables

- Evaluated to True or False
- Combine Boolean expression using and, or
- Flip Boolean value using not
- Membership: use in, not in



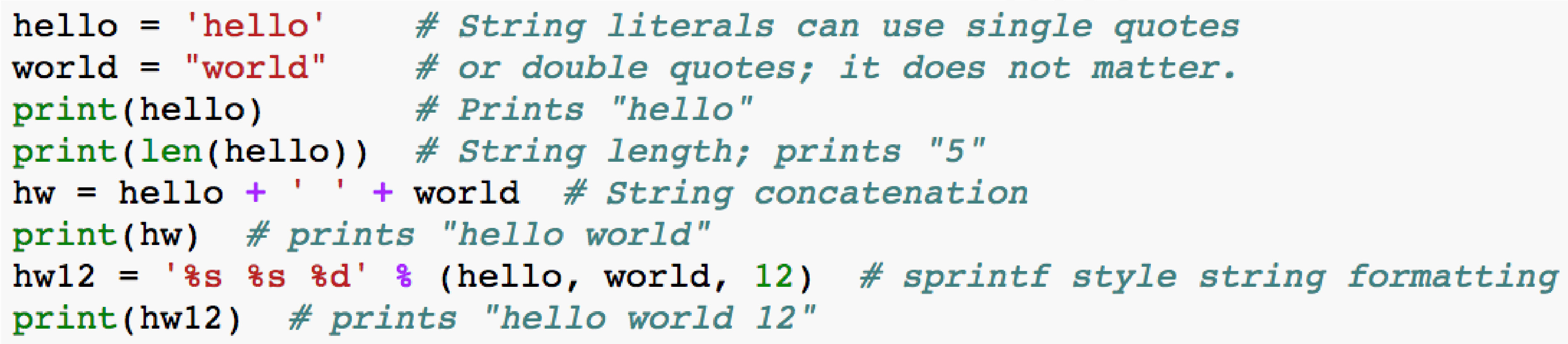
```
t = True
f = False
print(type(t)) # Prints "<class 'bool'>"
print(t and f) # Logical AND; prints "False"
print(t or f)  # Logical OR; prints "True"
print(not t)   # Logical NOT; prints "False"
```

Boolean Expression

- Status: True or False
- Comparison operators

Operators	Remarks	Example
==	Equal	`x == y`
!=	Not equal	`x != y`
>=	Greater than or equal to	`x >= y`
<=	Smaller than or equal to	`x <= y`
>	Greater than	`x > y`
<	Smaller than	`x < y`

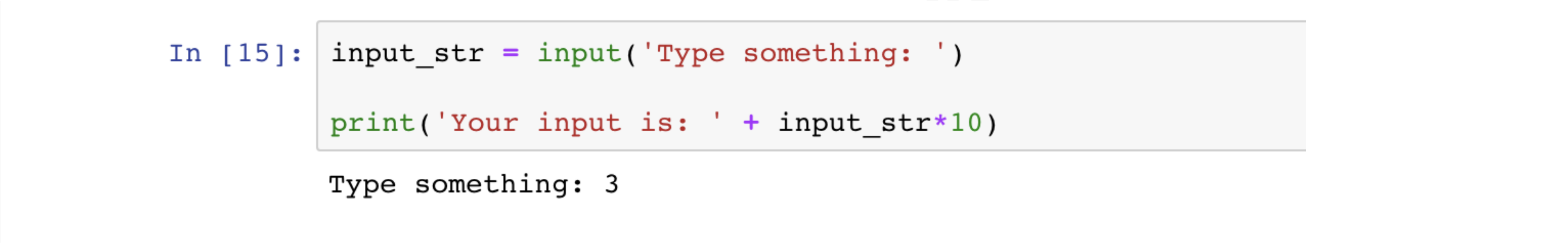
Strings



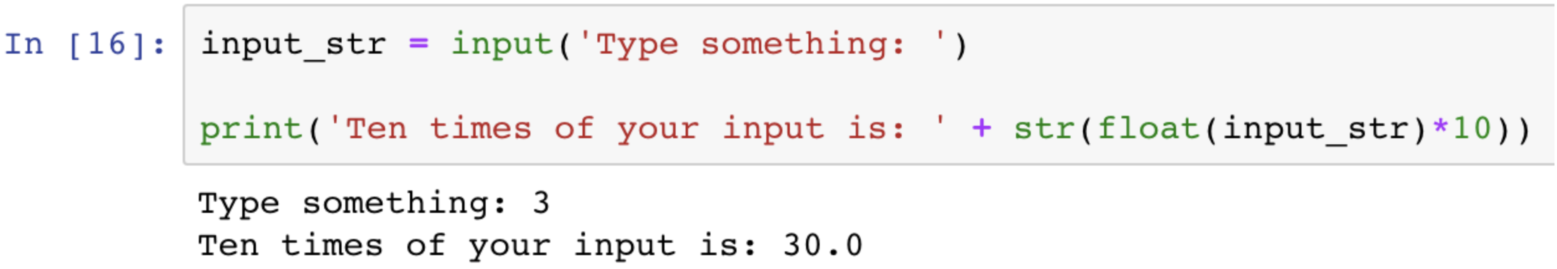
```
hello = 'hello'     # String literals can use single quotes
world = "world"     # or double quotes; it does not matter.
print(hello)        # Prints "hello"
print(len(hello))   # String length; prints "5"
hw = hello + ' ' + world  # String concatenation
print(hw)  # prints "hello world"
hw12 = '%s %s %d' % (hello, world, 12)  # sprintf style string formatting
print(hw12)  # prints "hello world 12"
```

Type Conversion

- Multiply a number by 10



```
In [15]: input_str = input('Type something: ')

         print('Your input is: ' + input_str*10)

Type something: 3
```



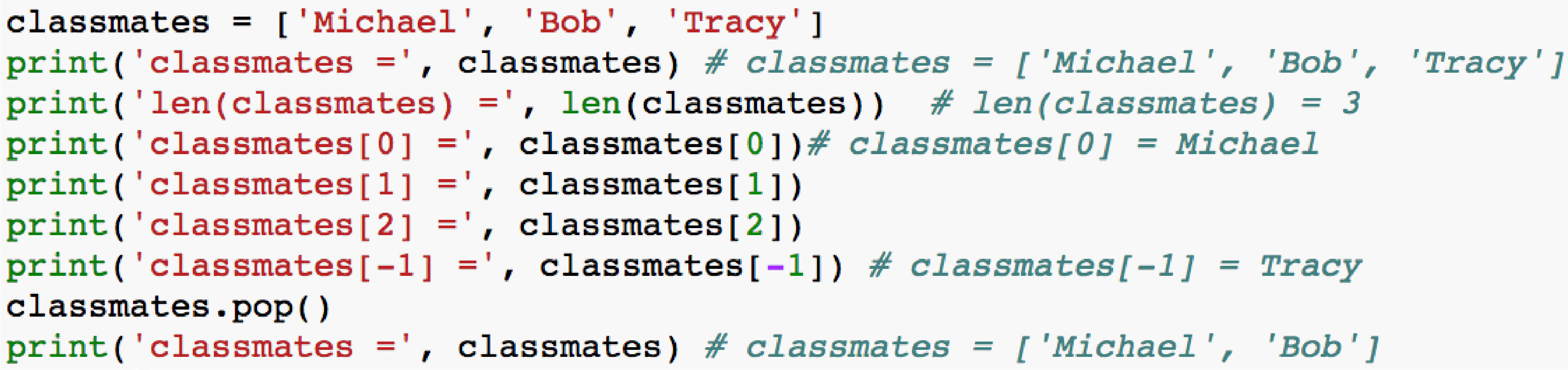
```
In [16]: input_str = input('Type something: ')

         print('Ten times of your input is: ' + str(float(input_str)*10))

Type something: 3
Ten times of your input is: 30.0
```

Compound Types

- List
  - e.g., [‘abcd’, 786, 2.23, ‘john’, 70.2]
- Tuple
  - e.g., (‘abcd’, 786, 2.23, ‘john’, 70.2)
- Dictionary
  - e.g., {‘cat’: ‘cute’, ‘dog’: ‘furry’}
- Set
  - e.g., {‘cat’, ‘dog’}

List

- A list is the Python equivalent of an array, but is resizeable and can contain elements of different types.
- Note that the first index is 0

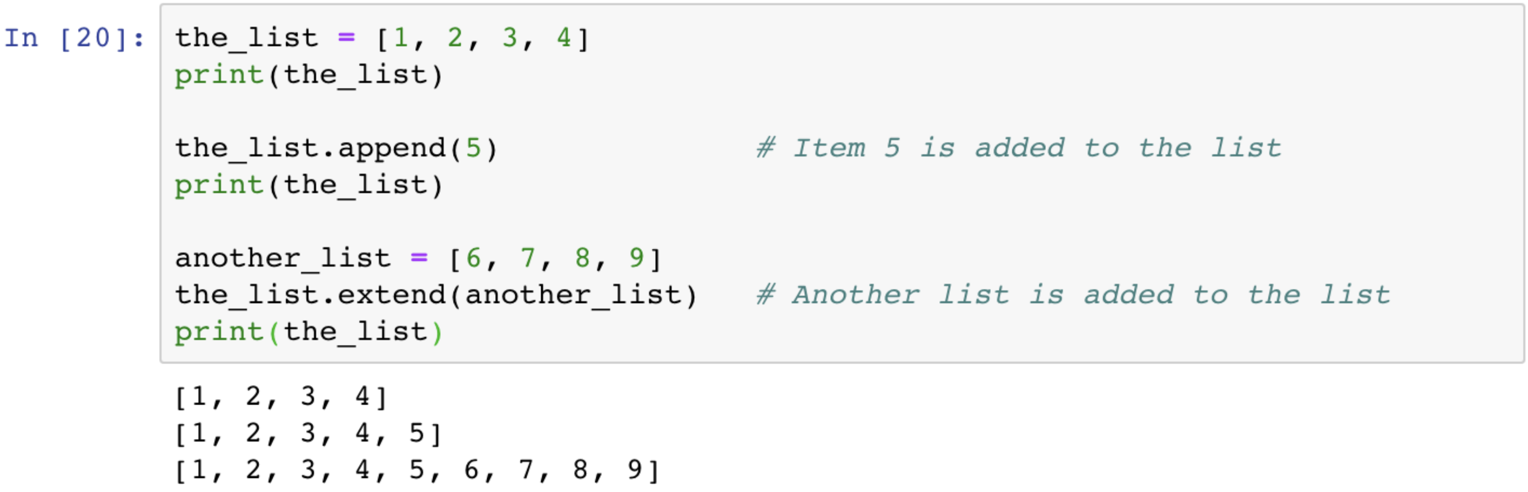


```
classmates = ['Michael', 'Bob', 'Tracy']
print('classmates =', classmates) # classmates = ['Michael', 'Bob', 'Tracy']
print('len(classmates) =', len(classmates))  # len(classmates) = 3
print('classmates[0] =', classmates[0])# classmates[0] = Michael
print('classmates[1] =', classmates[1])
print('classmates[2] =', classmates[2])
print('classmates[-1] =', classmates[-1]) # classmates[-1] = Tracy
classmates.pop()
print('classmates =', classmates) # classmates = ['Michael', 'Bob']
```

List

- Methods
  - Append, extend, insert, remove, pop, del

the_list	1	2	3	4
	0	1	2	3

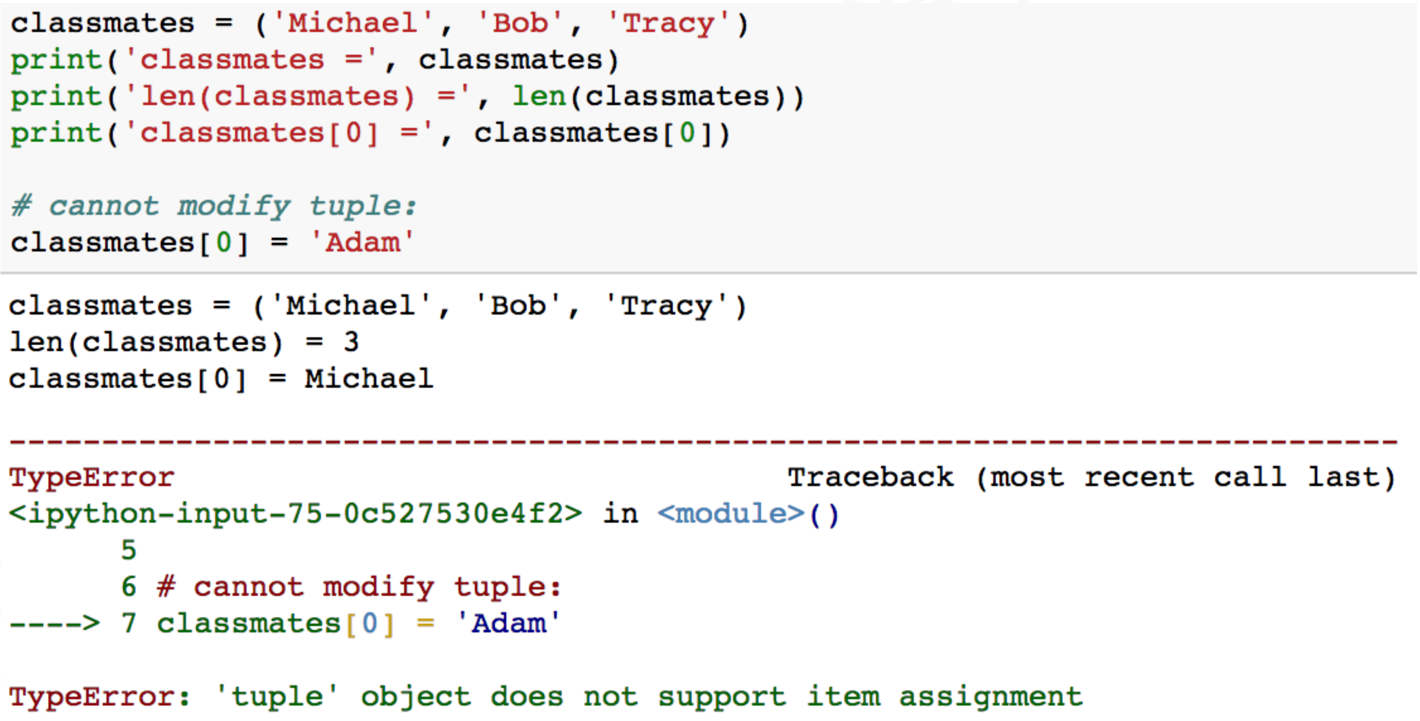


In [20]:
```
the_list = [1, 2, 3, 4]
print(the_list)

the_list.append(5)                   # Item 5 is added to the list
print(the_list)

another_list = [6, 7, 8, 9]
the_list.extend(another_list)    # Another list is added to the list
print(the_list)
```

```
[1, 2, 3, 4]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Tuple

- A tuple is in many ways similar to a list.
- But it is immutable ordered list of values.

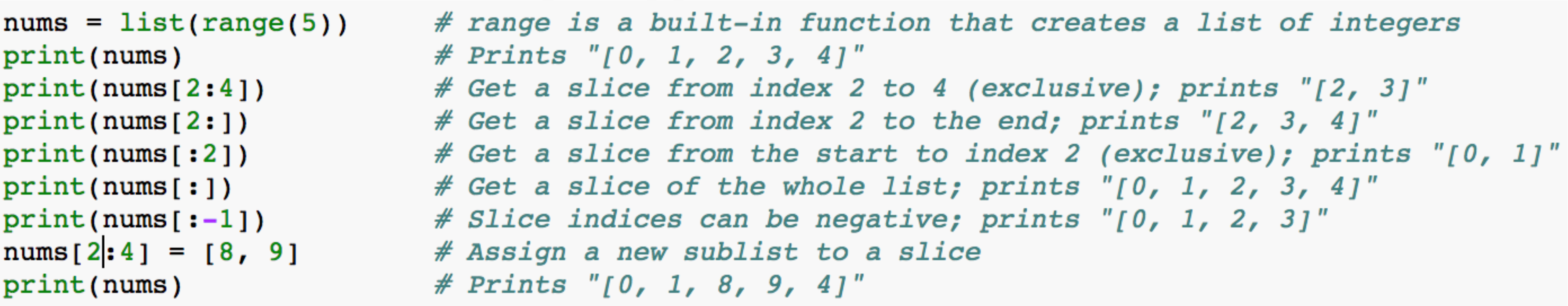


```
classmates = ('Michael', 'Bob', 'Tracy')
print('classmates =', classmates)
print('len(classmates) =', len(classmates))
print('classmates[0] =', classmates[0])

# cannot modify tuple:
classmates[0] = 'Adam'
```

```
classmates = ('Michael', 'Bob', 'Tracy')
len(classmates) = 3
classmates[0] = Michael

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-75-0c527530e4f2> in <module>()
      5 
      6 # cannot modify tuple:
----> 7 classmates[0] = 'Adam'

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

Slicing and Indexing

- Slicing: Python provides concise syntax to access sublists.
- Range: the range type represents an immutable sequence of numbers



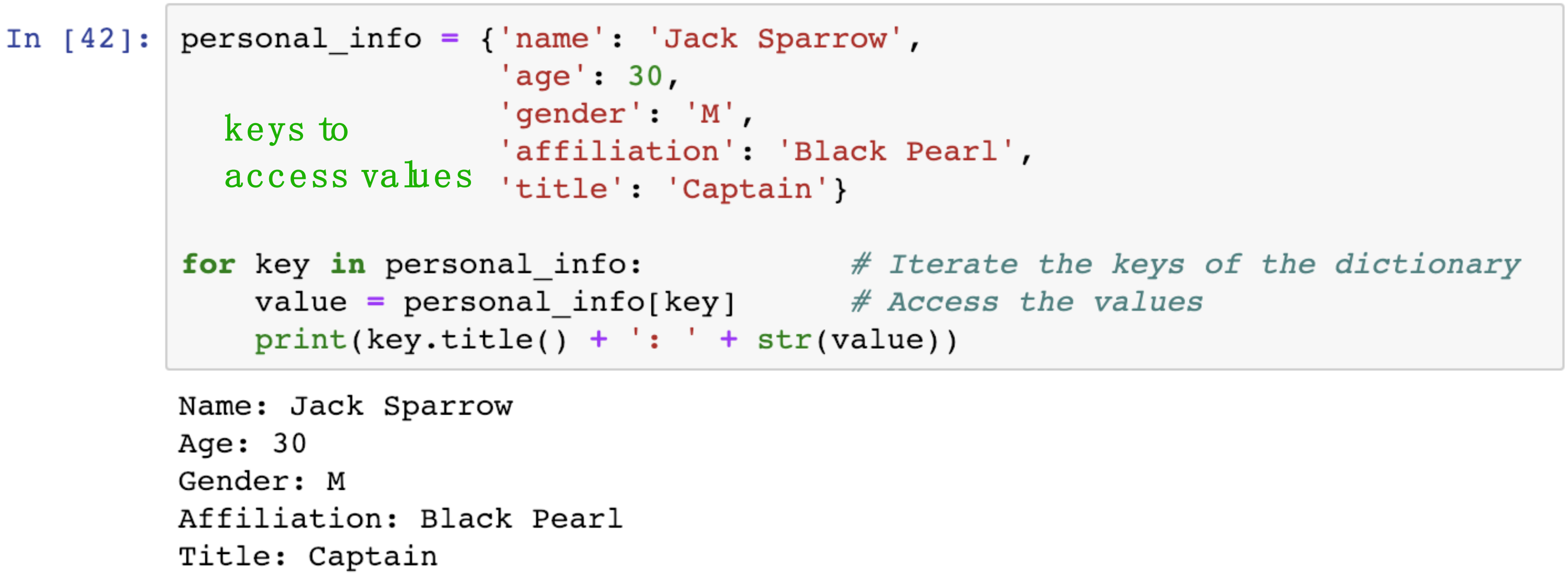
```
nums = list(range(5))     # range is a built-in function that creates a list of integers
print(nums)               # Prints "[0, 1, 2, 3, 4]"
print(nums[2:4])          # Get a slice from index 2 to 4 (exclusive); prints "[2, 3]"
print(nums[2:])           # Get a slice from index 2 to the end; prints "[2, 3, 4]"
print(nums[:2])           # Get a slice from the start to index 2 (exclusive); prints "[0, 1]"
print(nums[:])            # Get a slice of the whole list; prints "[0, 1, 2, 3, 4]"
print(nums[:-1])          # Slice indices can be negative; prints "[0, 1, 2, 3]"
nums[2:4] = [8, 9]        # Assign a new sublist to a slice
print(nums)               # Prints "[0, 1, 8, 9, 4]"
```

Dictionary

- A dictionary stores unordered (key, value) pairs.
- Search and Insert elements fast.
- key is immutable.
- More than one entry per key is not allowed.

Dictionary

- Example

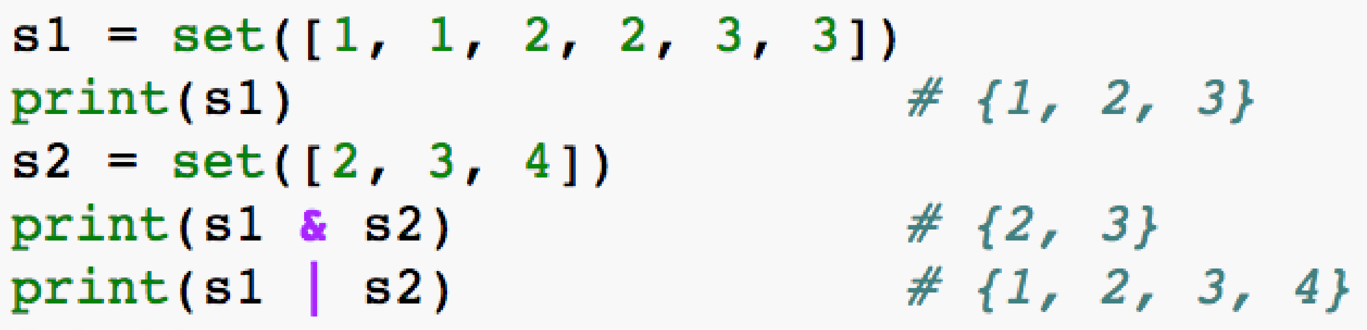


```
In [42]: personal_info = {'name': 'Jack Sparrow',
                          'age': 30,
                          'gender': 'M',
                          'affiliation': 'Black Pearl',
                          'title': 'Captain'}

         for key in personal_info:               # Iterate the keys of the dictionary
             value = personal_info[key]          # Access the values
             print(key.title() + ': ' + str(value))

         Name: Jack Sparrow
         Age: 30
         Gender: M
         Affiliation: Black Pearl
         Title: Captain
```

keys to
access values

Set

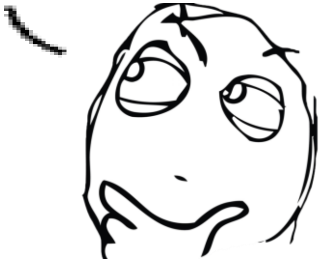
- A set is an unordered collection of distinct elements (same as the corresponding concepts in math)



```
s1 = set([1, 1, 2, 2, 3, 3])
print(s1)                           # {1, 2, 3}
s2 = set([2, 3, 4])
print(s1 & s2)                      # {2, 3}
print(s1 | s2)                      # {1, 2, 3, 4}
```

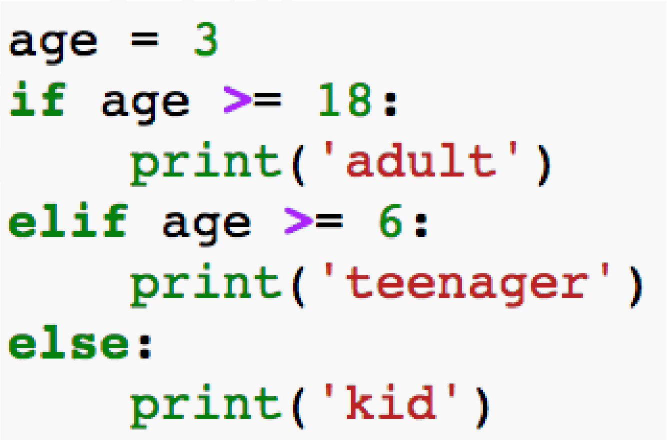
Python Control Flow

Anyone wanna go for lunch? If not, I'll ask again later.



Conditional Statement

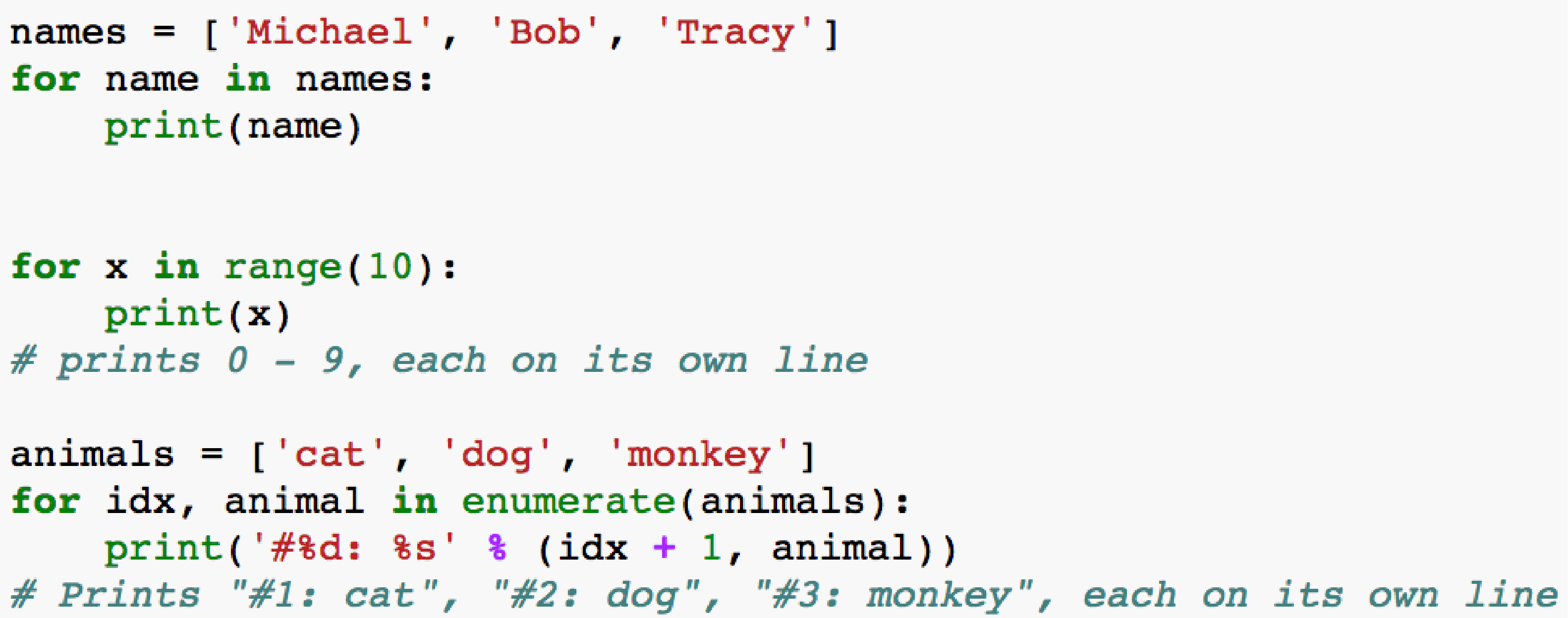
- Test if a Boolean expression is True or False and run different code in each case
- Can split the code into more than two cases



```
age = 3
if age >= 18:
    print('adult')
elif age >= 6:
    print('teenager')
else:
    print('kid')
```

For Loops

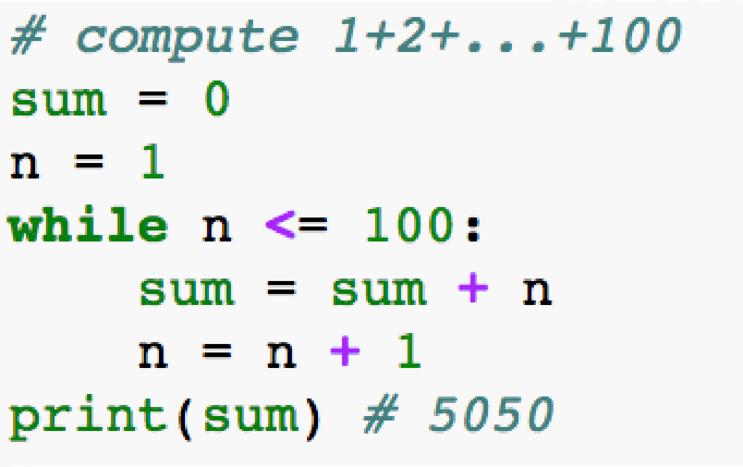
- Examples



```
names = ['Michael', 'Bob', 'Tracy']
for name in names:
    print(name)


for x in range(10):
    print(x)
# prints 0 - 9, each on its own line

animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
    print('#%d: %s' % (idx + 1, animal))
# Prints "#1: cat", "#2: dog", "#3: monkey", each on its own line
```

While Loops

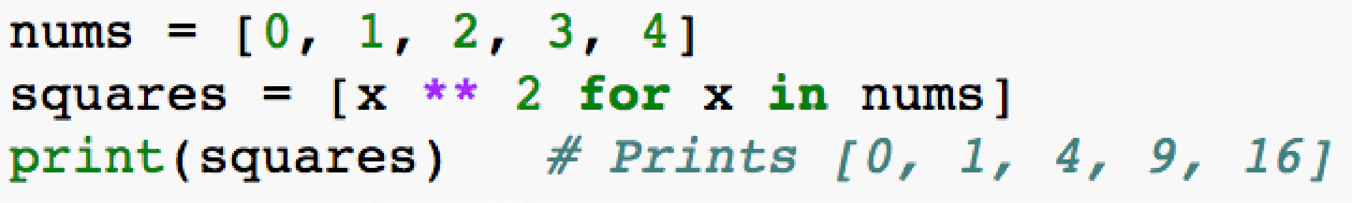
- You can also use while to loop



```
# compute 1+2+...+100
sum = 0
n = 1
while n <= 100:
    sum = sum + n
    n = n + 1
print(sum) # 5050
```

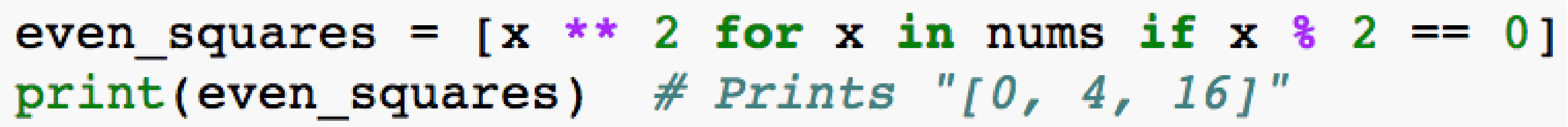
List Comprehension

- To transform one list of data into another easily. Instead of using loops, we can write



```python
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares)   # Prints [0, 1, 4, 9, 16]
```

- You can add conditional control

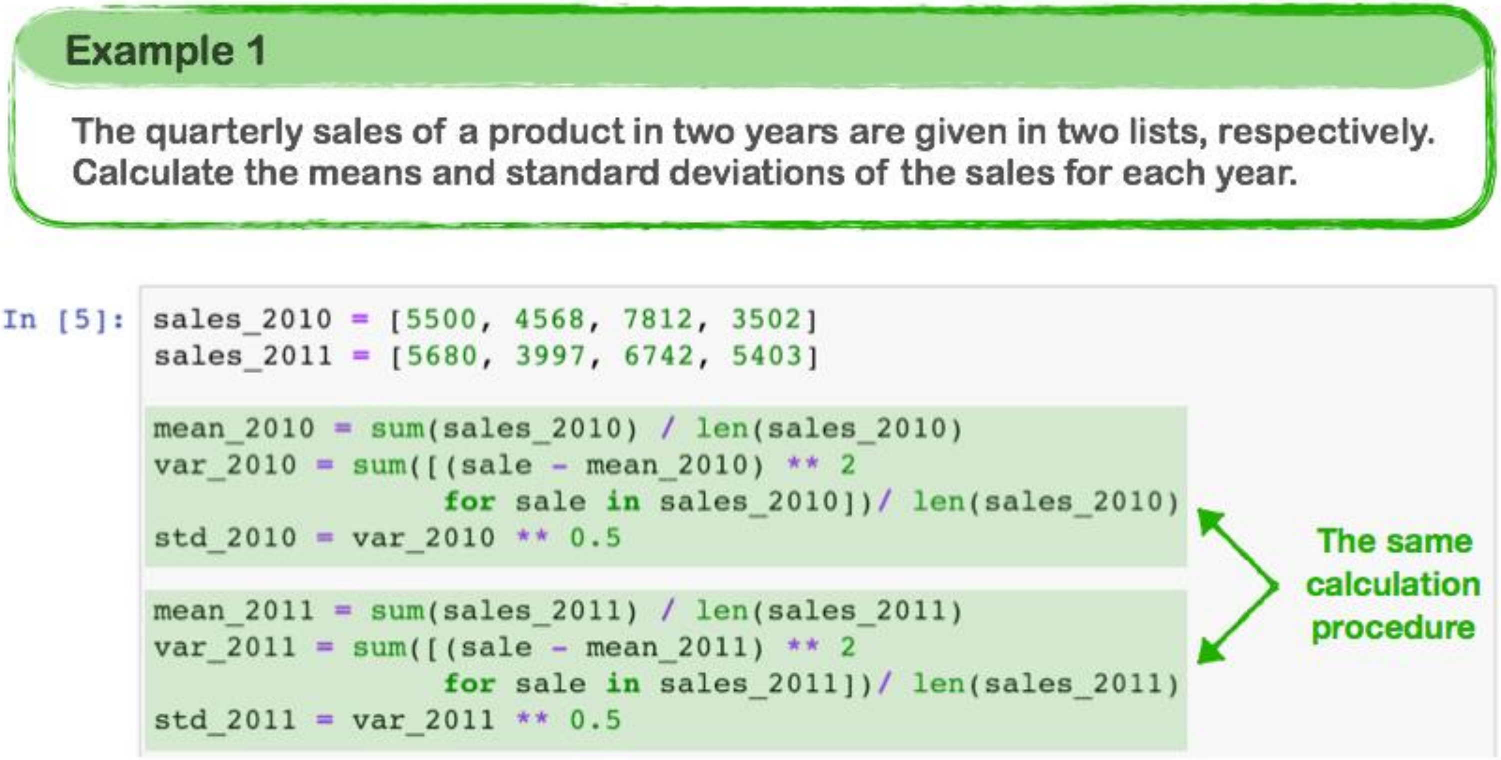


```python
even_squares = [x ** 2 for x in nums if x % 2 == 0]
print(even_squares)  # Prints "[0, 4, 16]"
```

Functions

Functions

- Why needed?



Example 1

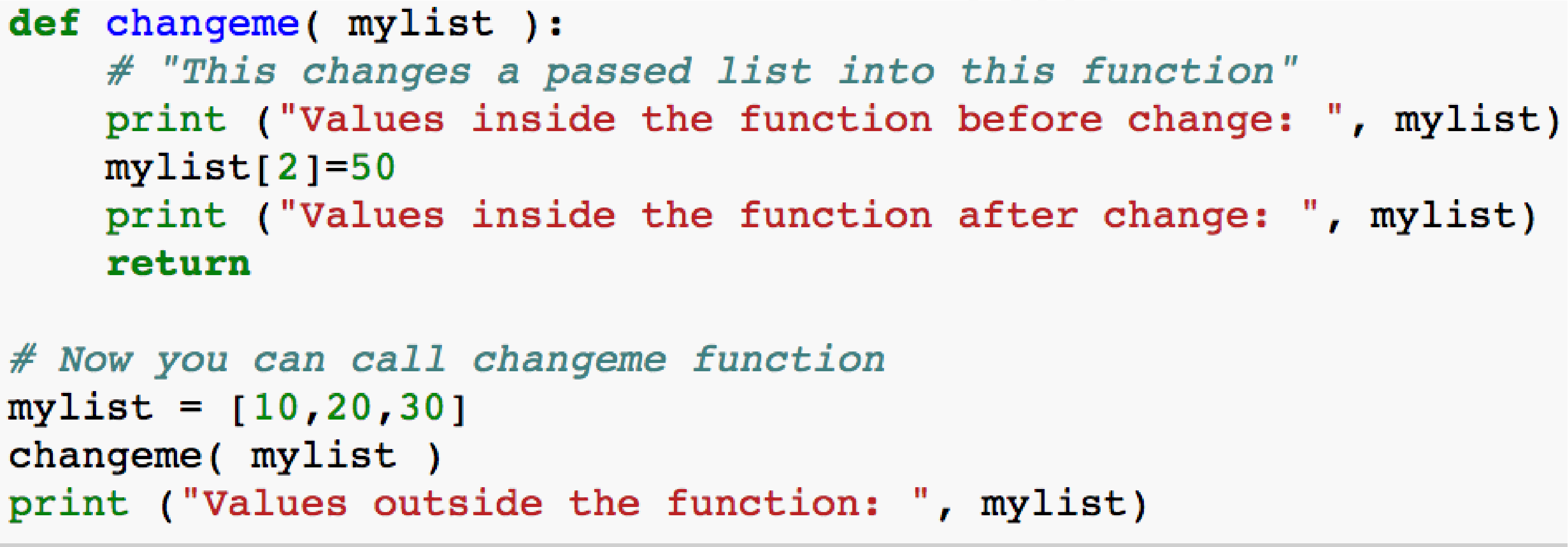
The quarterly sales of a product in two years are given in two lists, respectively. Calculate the means and standard deviations of the sales for each year.

```
In [5]: sales_2010 = [5500, 4568, 7812, 3502]
        sales_2011 = [5680, 3997, 6742, 5403]

        mean_2010 = sum(sales_2010) / len(sales_2010)
        var_2010 = sum([(sale - mean_2010) ** 2
                        for sale in sales_2010])/ len(sales_2010)
        std_2010 = var_2010 ** 0.5

        mean_2011 = sum(sales_2011) / len(sales_2011)
        var_2011 = sum([(sale - mean_2011) ** 2
                        for sale in sales_2011])/ len(sales_2011)
        std_2011 = var_2011 ** 0.5
```

The same calculation procedure

Passing by Reference

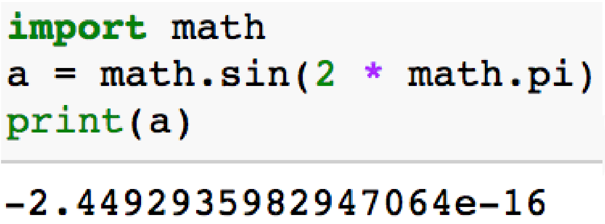
- All parameters (arguments) in Python are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function.



```
def changeme( mylist ):
    # "This changes a passed list into this function"
    print ("Values inside the function before change: ", mylist)
    mylist[2]=50
    print ("Values inside the function after change: ", mylist)
    return

# Now you can call changeme function
mylist = [10,20,30]
changeme( mylist )
print ("Values outside the function: ", mylist)
```

Function from Modules

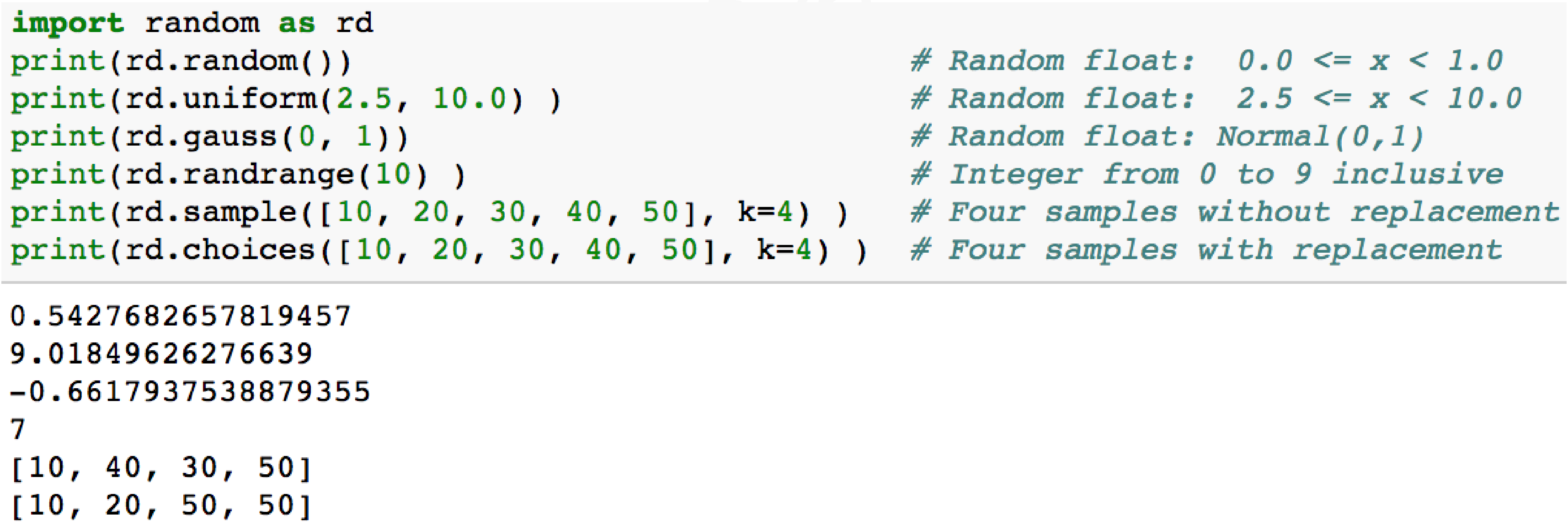
- math



```
import math
a = math.sin(2 * math.pi)
print(a)
```

```
-2.4492935982947064e-16
```

- random



```
import random as rd
print(rd.random())                              # Random float:  0.0 <= x < 1.0
print(rd.uniform(2.5, 10.0) )                   # Random float:  2.5 <= x < 10.0
print(rd.gauss(0, 1))                           # Random float: Normal(0,1)
print(rd.randrange(10) )                        # Integer from 0 to 9 inclusive
print(rd.sample([10, 20, 30, 40, 50], k=4) )    # Four samples without replacement
print(rd.choices([10, 20, 30, 40, 50], k=4) )   # Four samples with replacement
```

```
0.5427682657819457
9.01849626276639
-0.6617937538879355
7
[10, 40, 30, 50]
[10, 20, 50, 50]
```

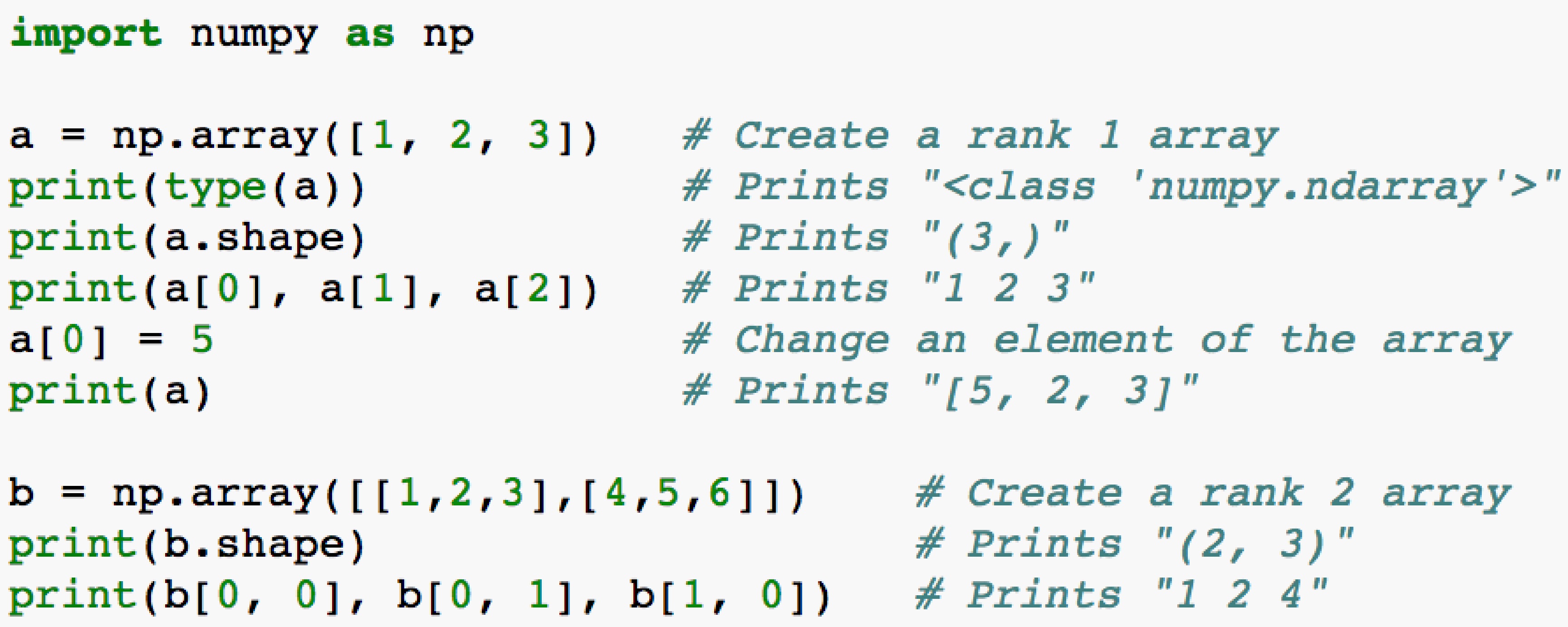
NumPy Module

- Vector, Matrix, and Array

- For Linear Algebra use, you need to install and import numpy module.
  - Slicing and transpose
  - Basic vector and matrix operation
  - Basic Linear Algebra

NumPy Module

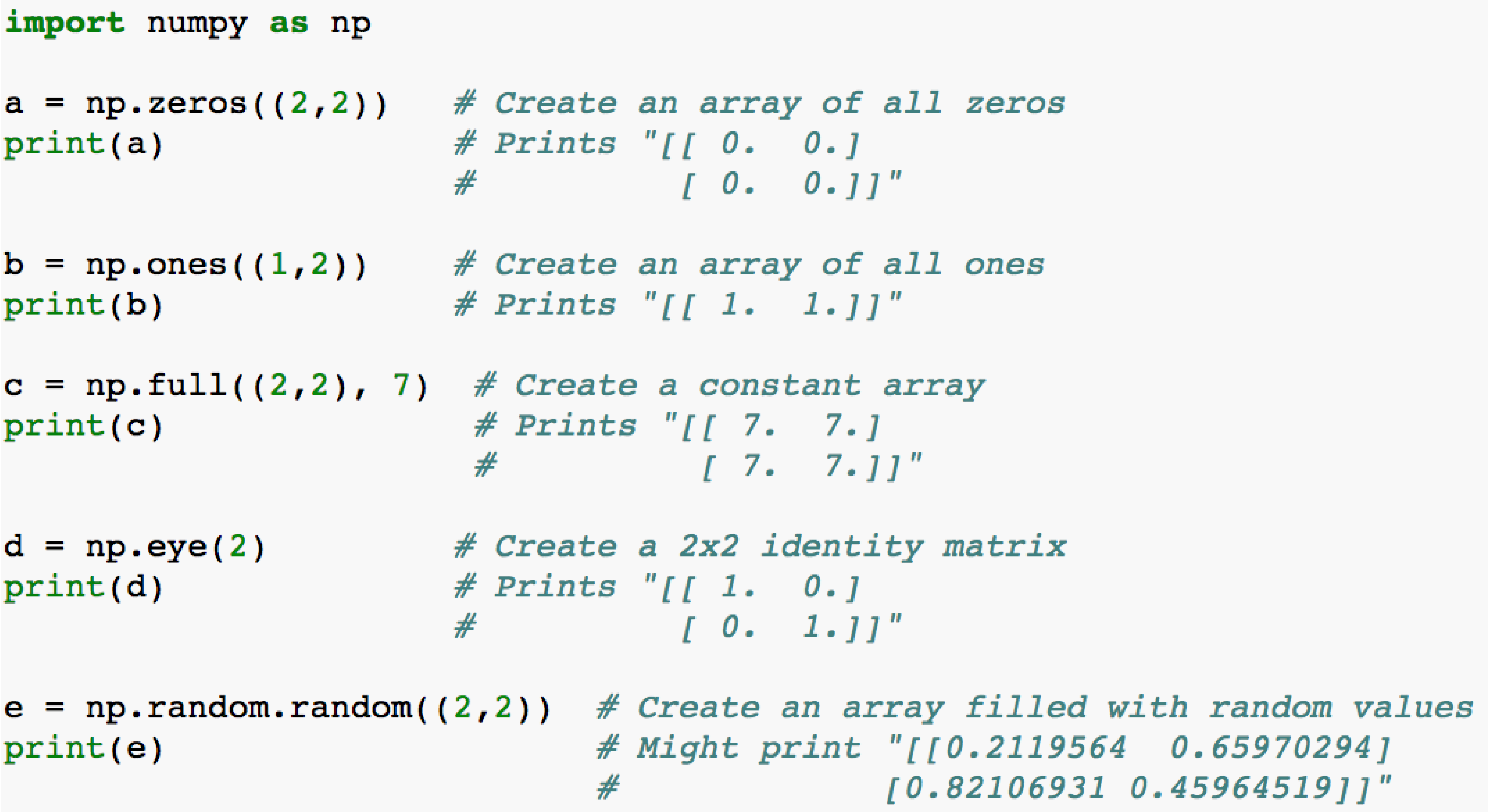
- Basic Initialization



```python
import numpy as np

a = np.array([1, 2, 3])   # Create a rank 1 array
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Change an element of the array
print(a)                  # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array
print(b.shape)                     # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```

NumPy Module

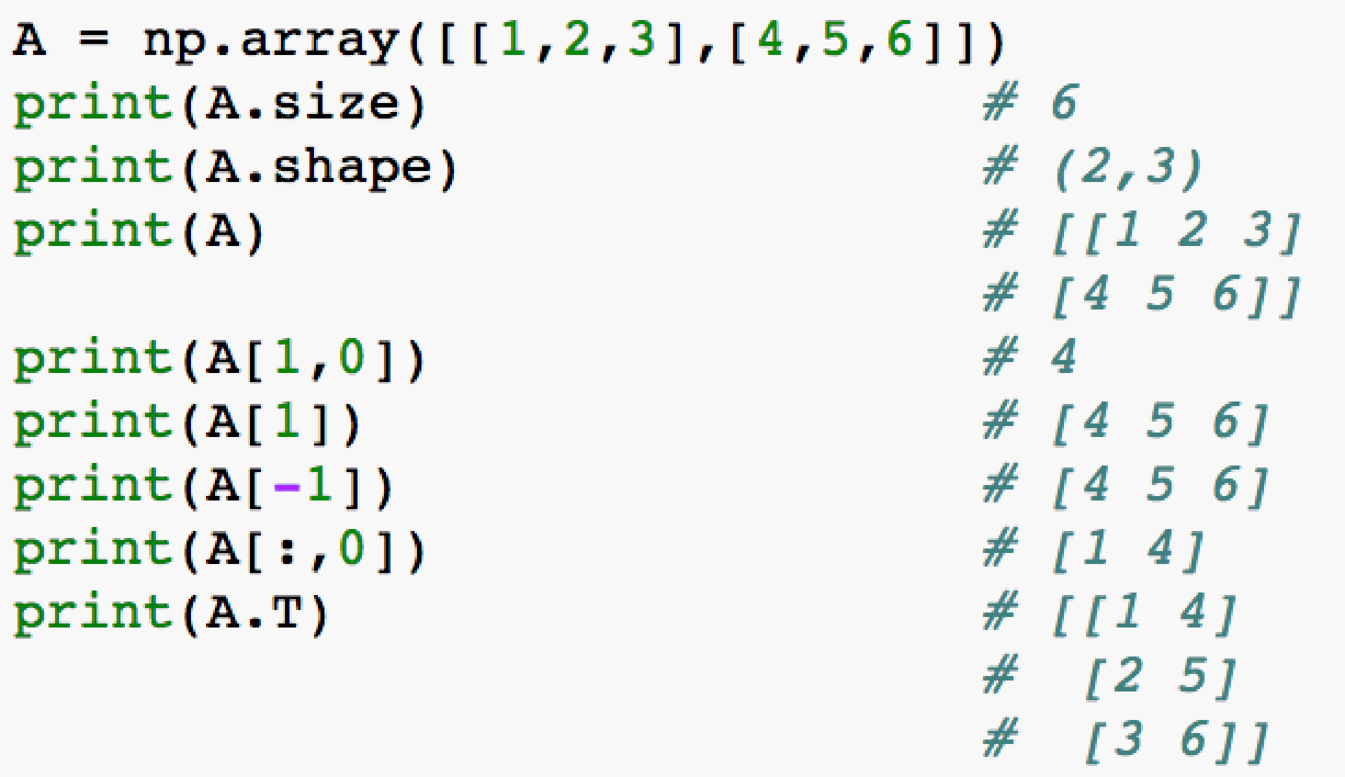
- Special Initialization



```
import numpy as np

a = np.zeros((2,2))   # Create an array of all zeros
print(a)              # Prints "[[ 0.  0.]
                      #          [ 0.  0.]]"

b = np.ones((1,2))    # Create an array of all ones
print(b)              # Prints "[[ 1.  1.]]"

c = np.full((2,2), 7)  # Create a constant array
print(c)               # Prints "[[ 7.  7.]
                       #          [ 7.  7.]]"

d = np.eye(2)         # Create a 2x2 identity matrix
print(d)              # Prints "[[ 1.  0.]
                      #          [ 0.  1.]]"

e = np.random.random((2,2))  # Create an array filled with random values
print(e)                     # Might print "[[0.2119564  0.65970294]
                             #               [0.82106931 0.45964519]]"
```

NumPy Module

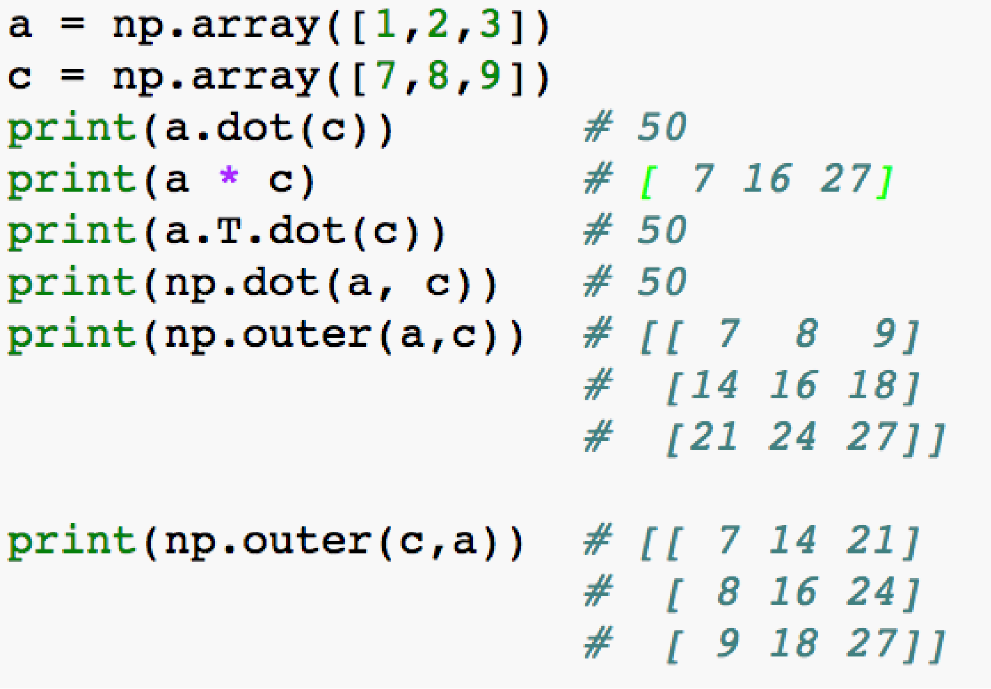
- Slicing and Transpose



```
A = np.array([[1,2,3],[4,5,6]])
print(A.size)                 # 6
print(A.shape)                # (2,3)
print(A)                      # [[1 2 3]
                              # [4 5 6]]
print(A[1,0])                 # 4
print(A[1])                   # [4 5 6]
print(A[-1])                  # [4 5 6]
print(A[:,0])                 # [1 4]
print(A.T)                    # [[1 4]
                              #  [2 5]
                              #  [3 6]]
```

NumPy Module

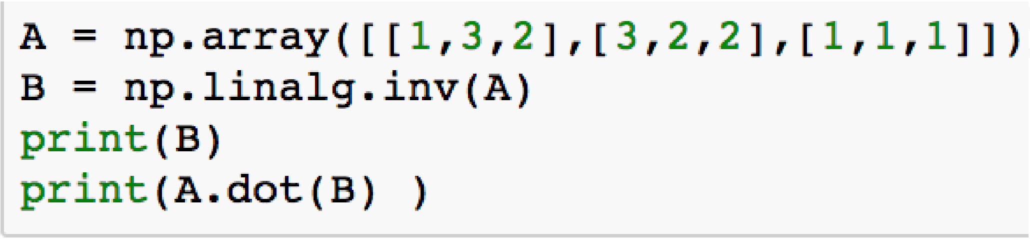
- Vector and Matrix Multiplication
  - n-dim vectors are NOT 1xn or nx1 matrices and can be either column or row vectors



```
a = np.array([1,2,3])
c = np.array([7,8,9])
print(a.dot(c))        # 50
print(a * c)           # [ 7 16 27]
print(a.T.dot(c))      # 50
print(np.dot(a, c))    # 50
print(np.outer(a,c))   # [[ 7  8  9]
                       #  [14 16 18]
                       #  [21 24 27]]

print(np.outer(c,a))   # [[ 7 14 21]
                       #  [ 8 16 24]
                       #  [ 9 18 27]]
```

NumPy Module

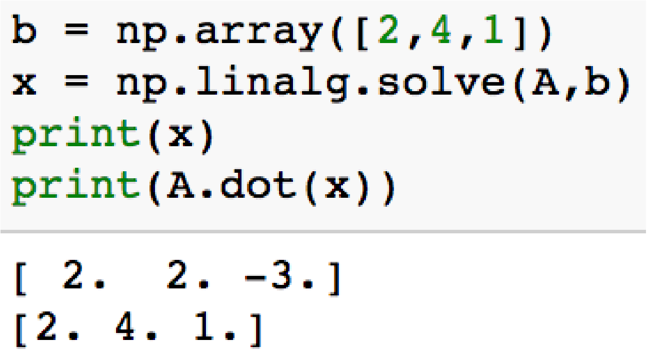
- Matrix Inverse



```
A = np.array([[1,3,2],[3,2,2],[1,1,1]])
B = np.linalg.inv(A)
print(B)
print(A.dot(B) )
```

```
[[ 0.   1. -2.]
 [ 1.   1. -4.]
 [-1. -2.   7.]]
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

- Solve linear systems

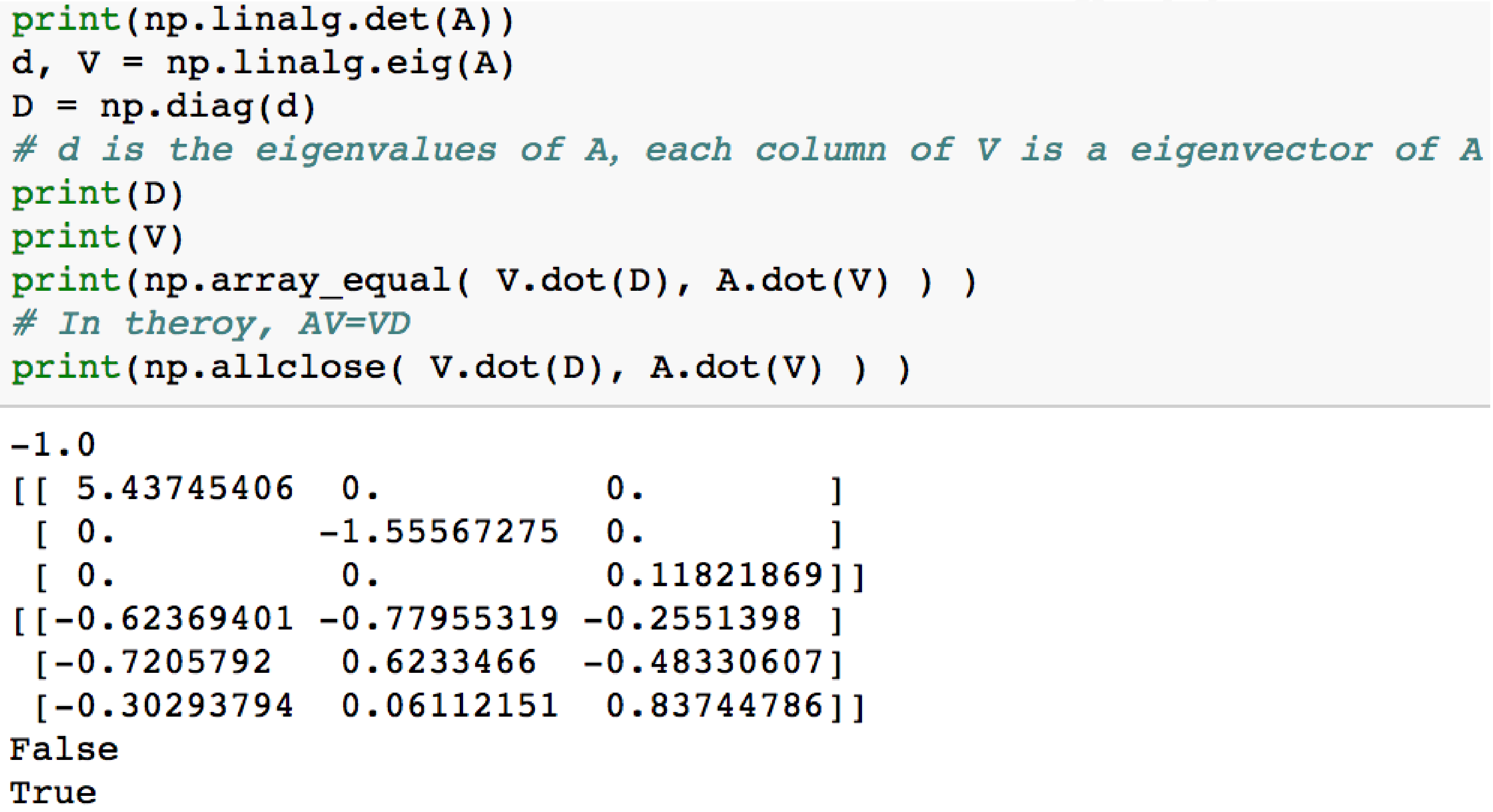


```
b = np.array([2,4,1])
x = np.linalg.solve(A,b)
print(x)
print(A.dot(x))
```

```
[ 2.  2. -3.]
[2. 4. 1.]
```

NumPy Module

- Get determinant, eigen-values and -vectors



```python
print(np.linalg.det(A))
d, V = np.linalg.eig(A)
D = np.diag(d)
# d is the eigenvalues of A, each column of V is a eigenvector of A
print(D)
print(V)
print(np.array_equal( V.dot(D), A.dot(V) ) )
# In theroy, AV=VD
print(np.allclose( V.dot(D), A.dot(V) ) )
```

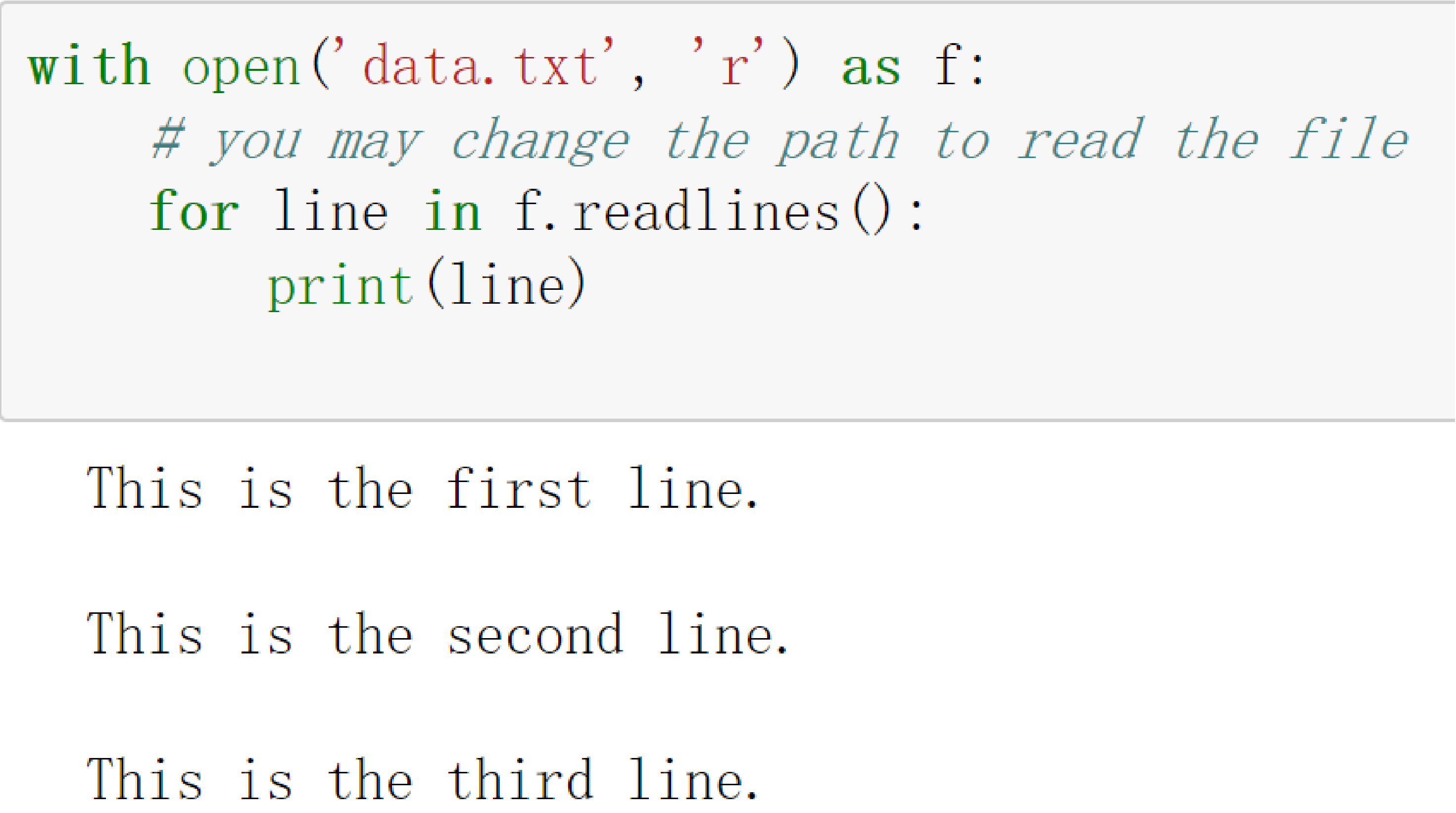
```
-1.0
[[ 5.43745406  0.          0.        ]
 [ 0.         -1.55567275  0.        ]
 [ 0.          0.          0.11821869]]
[[-0.62369401 -0.77955319 -0.2551398 ]
 [-0.7205792   0.6233466  -0.48330607]
 [-0.30293794  0.06112151  0.83744786]]
False
True
```

File Input/Output

- First, open a file by open()
- Then, read the file:
  - For small file, we can use read() to read the file in a batch
  - For large file, we can use read(size)
  - readline() read a line each time
- Finally, use close() to close the file

File Input/Output

- Example

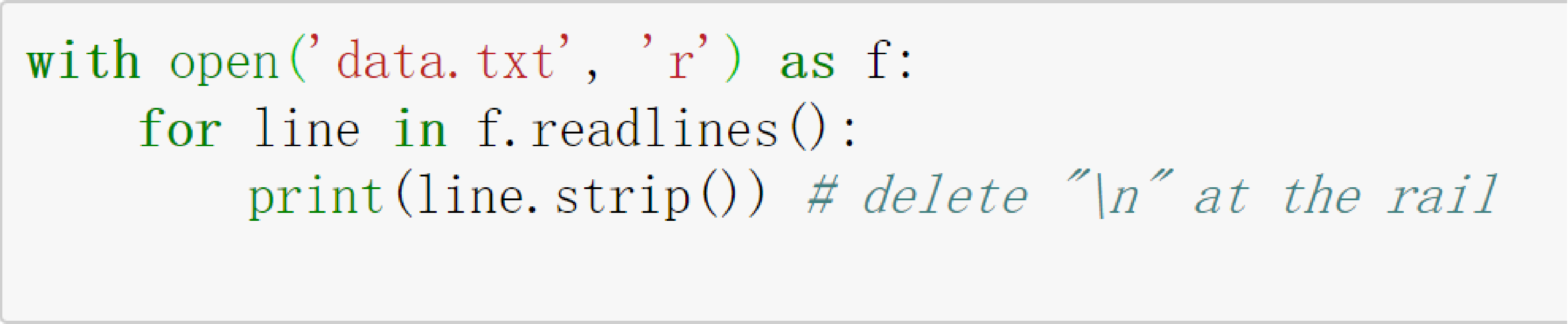


```python
with open(’data.txt’, ’r’) as f:
    # you may change the path to read the file
    for line in f.readlines():
        print(line)
```

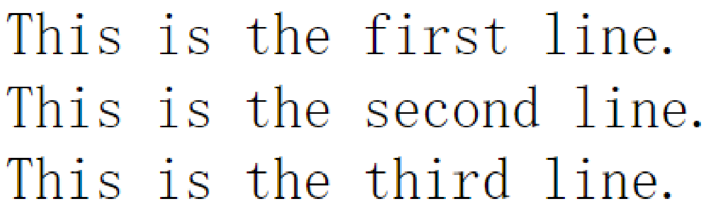
```
This is the first line.

This is the second line.

This is the third line.
```

File Input/Output

- Using the with keyword when dealing with file objects is highly suggested.



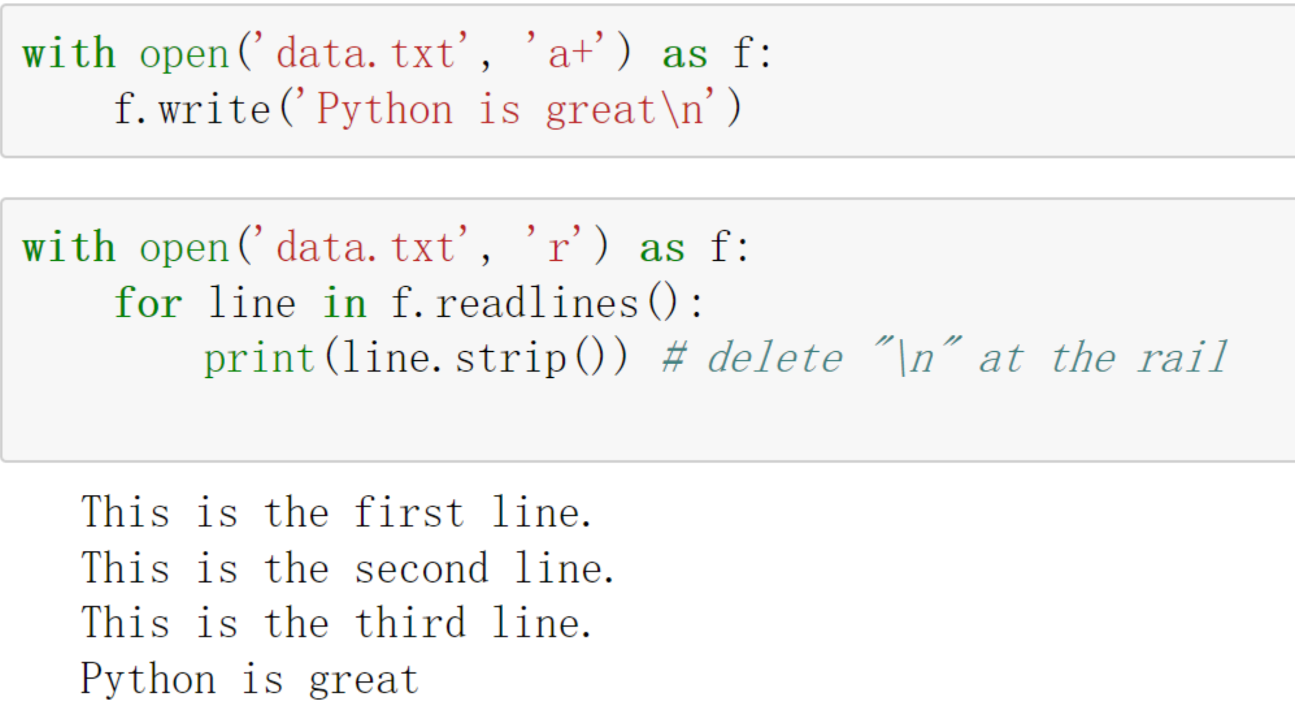
```
with open(’data.txt’, ’r’) as f:
    for line in f.readlines():
        print(line.strip()) # delete ”\n” at the rail
```



```
This is the first line.
This is the second line.
This is the third line.
```

File Input/Output

- Add contents at the end of the file. Notice the argument is a ‘a+’.
- If you want to rewrite the file from the beginning, use ‘w’ or ‘r+’.



```
with open(’data.txt’, ’a+’) as f:
    f.write(’Python is great\n’)
```

```
with open(’data.txt’, ’r’) as f:
    for line in f.readlines():
        print(line.strip()) # delete ″\n″ at the rail
```

```
This is the first line.
This is the second line.
This is the third line.
Python is great
```