Ton Duc Thang University
Faculty of Information Technology

Applied Calculus for IT - 501031
Python tutorial 02

1. Python Indentation

Indentation refers to the spaces at the beginning of a code line. The indentation in Python is very important, and it indicate a block of code.

- Example:

```
if 5 > 2:
  print("Five is greater than two!")
```

- Python will give you an error if you skip the indentation:

```
if 5 > 2:
print("Five is greater than two!")
```

The content of error is: *IndentationError: expected an indented block*

- The number of spaces is up to you as a programmer, but it has to be at least one.

```
if 5 > 2:
 print("Five is greater than two!")
if 5 > 2:
        print("Five is greater than two!")
```

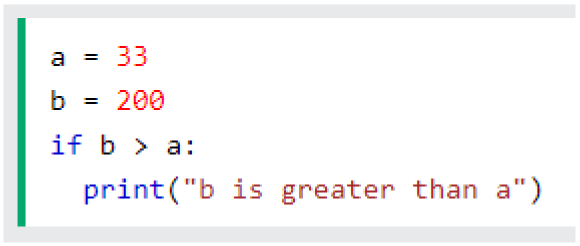
- You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:

```
if 5 > 2:
 print("Five is greater than two!")
        print("Five is greater than two!")
```

The content of error is: *IndentationError: unexpected indent*

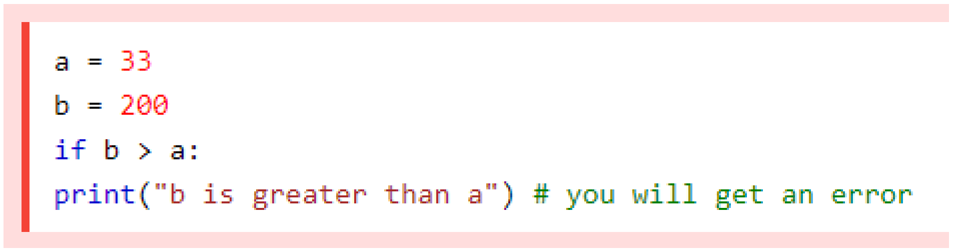
2. Python Conditions and If statements

- Python supports the usual logical conditions from mathematics, for ex., a == b, a != b, a > b, … These conditions can be used in several ways, most commonly in "if statements" and loops.



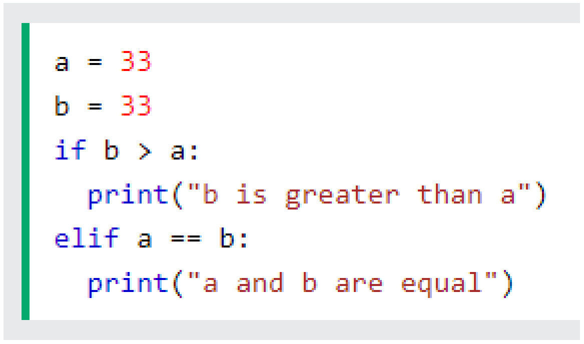
```python
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

- Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. If statement, without indentation (will raise an error):



```python
a = 33
b = 200
if b > a:
print("b is greater than a") # you will get an error
```

- The **elif** keyword is pythons way of saying "if the previous conditions were not true, then try this condition":



```python
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

- The **else** keyword catches anything which isn't caught by the preceding conditions:

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

- The **and/or** keyword is a logical operator, and is used to combine conditional statements:

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

- You can have if statements inside if statements, this is called nested if statements:

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

3. Python Functions

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

To call a function, use the function name followed by parenthesis, and then parameters can be passed inside the parentheses.

print(“sum of a and b:”)
range(1, 10, 3)

4. Python Lists

- Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.
- Lists are created using square brackets:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

- List items are ordered, changeable, and allow duplicate values.
- To determine how many items a list has, use the **len**() function:

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

- A list can contain different data types:

```
list1 = ["abc", 34, True, 40, "male"]
```

- List items are indexed and you can access them by referring to the index number (the first item has index **[0]**, the second item has index [1]):

Print the second item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

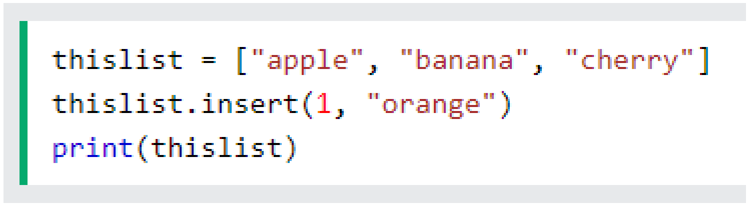
Change the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

- To add an item to the end of the list, use the **append**() method:

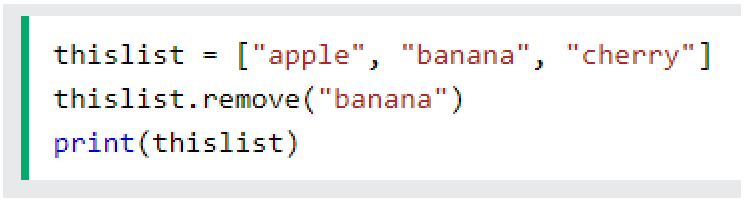
```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

- To insert a list item at a specified index, use the **insert**() method:



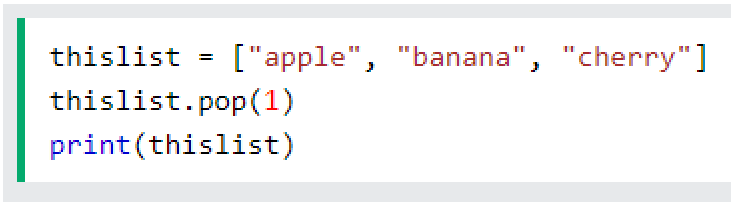
```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

- The **remove**() method removes the specified item:



```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

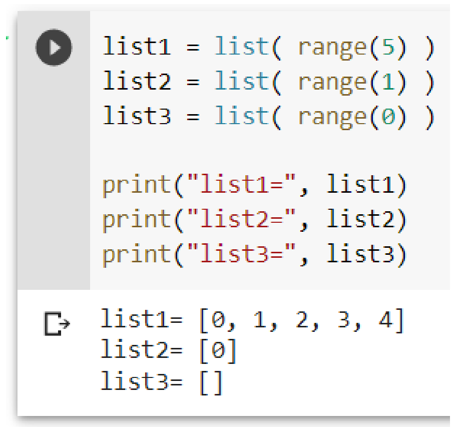
- The **pop**() method removes the specified index:



```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

- range() function: returns a sequence of **integer numbers**, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.
  - Syntax: `range(start, stop, step)`
  - Parameter:
    - *start*: [optional] start value of the sequence, default is 0
    - *stop*: next value after the end value of the sequence
    - *step*: [optional] denoting the difference between any two numbers in the sequence, default is 1
  - Return: Returns a range type object.

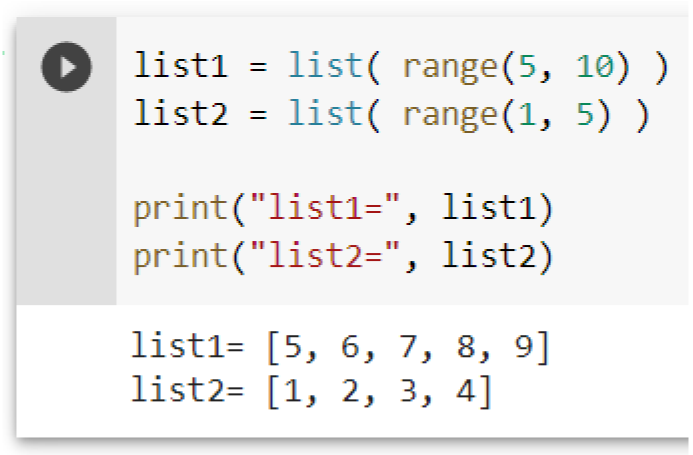
  - 1 parameter form: `range(stop)`



```
list1 = list( range(5) )
list2 = list( range(1) )
list3 = list( range(0) )

print("list1=", list1)
print("list2=", list2)
print("list3=", list3)
```

```
list1= [0, 1, 2, 3, 4]
list2= [0]
list3= []
```

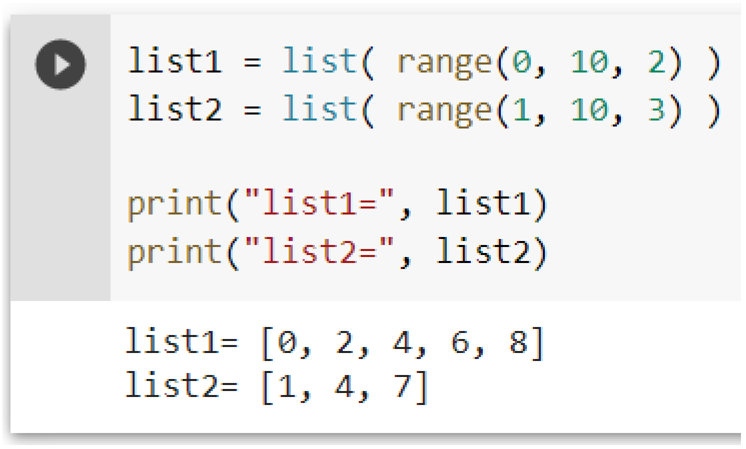
- 2 parameter form: `range(start, stop)`



```
list1 = list( range(5, 10) )
list2 = list( range(1, 5) )

print("list1=", list1)
print("list2=", list2)
```

```
list1= [5, 6, 7, 8, 9]
list2= [1, 2, 3, 4]
```

- 3 parameter form: `range(start, stop, step)`

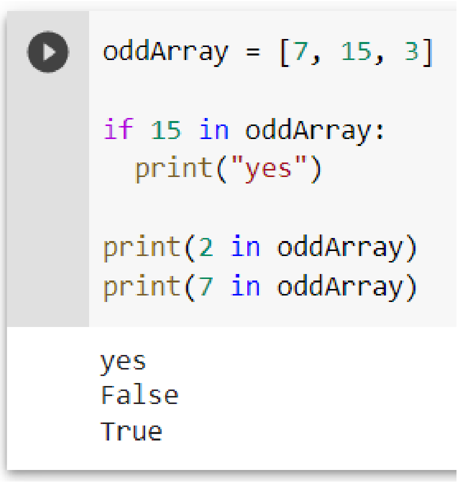


```
list1 = list( range(0, 10, 2) )
list2 = list( range(1, 10, 3) )

print("list1=", list1)
print("list2=", list2)
```

```
list1= [0, 2, 4, 6, 8]
list2= [1, 4, 7]
```

https://cs.stanford.edu/people/nick/py/python-range.html

- **in** Keyword
  - used to check if a value is present in a sequence (list, range, string etc.).
  - used to iterate through a sequence in a **for loop**

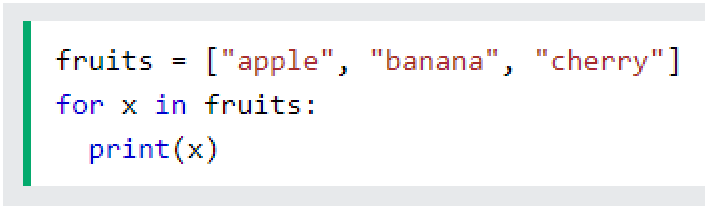


```
oddArray = [7, 15, 3]

if 15 in oddArray:
  print("yes")

print(2 in oddArray)
print(7 in oddArray)
```

```
yes
False
True
```

5. Python for loops

- A **for** loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string):



```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

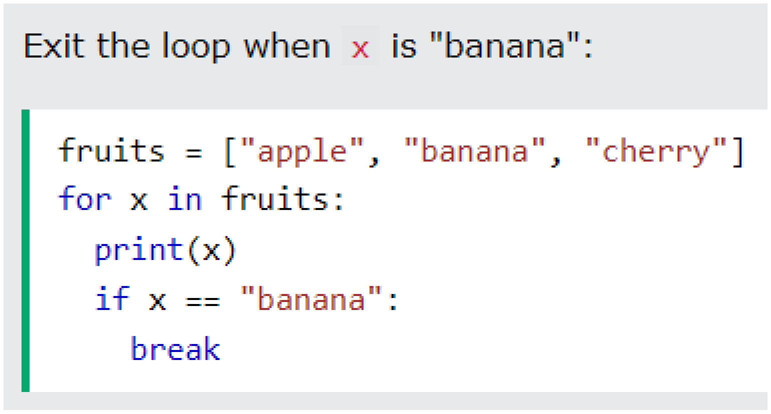
```
arr = [7, 3, 2, 10]
for x in arr:
  print(x)

for i in range( len(arr) ):
  print( arr[i] )
```

- To loop through a set of code a specified number of times, we can use the range() function:

```
for i in range(5):
  print("Hello world!")
```

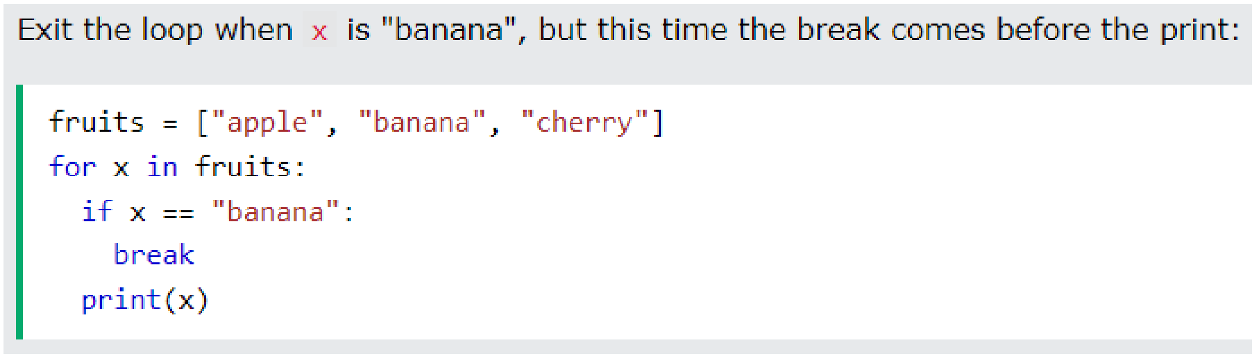
- The **break** statement

  With the break statement we can stop the loop before it has looped through all the items:



Exit the loop when x is "banana":

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

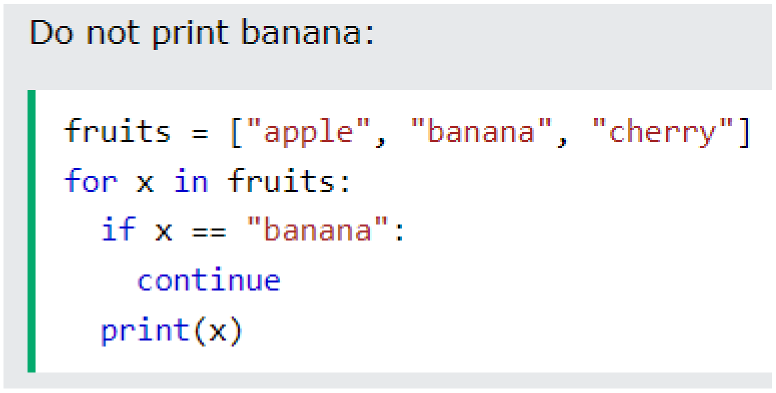


Exit the loop when x is "banana", but this time the break comes before the print:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

- The **continue** statement

  With the continue statement we can stop the current iteration of the loop, and continue with the next:

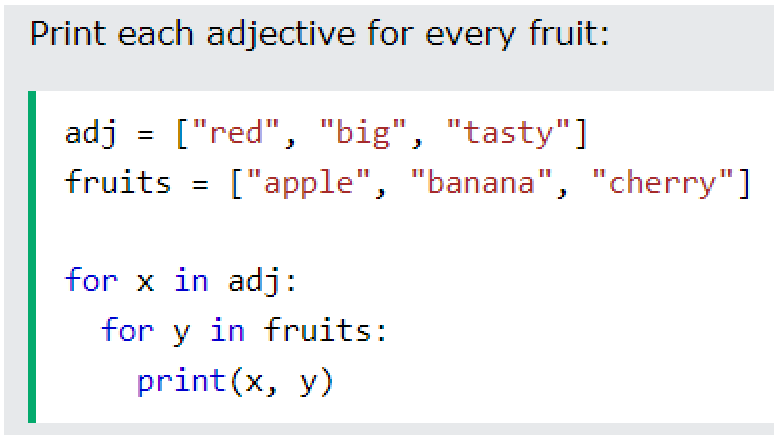


Do not print banana:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

- Nested Loops

  A nested loop is a loop inside a loop. The "inner loop" will be executed one time for each iteration of the "outer loop":



Print each adjective for every fruit:

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

- **Practice examples:**

```
#Print each even number in the array
arr = [7, 3, 2, 10, 5, 0]
for x in arr:
  if x % 2 == 0:
    print(x)

#Print each even number between 0 and 10 (both included)
for x in range(11):
  if x % 2 == 0:
    print("ex2", x)
```

Exercise 1

Write a Python program to print odd numbers between 50 and 100.

Exercise 2

Write a Python program to print the integer numbers which are divisible by 7 and multiple of 5, between 1500 and 2700 (both included).

Exercise 3

Write a Python program to prints all the integer numbers from 0 to 20 except 3 and 16. Note : Use 'continue' statement.

Exercise 4

Write a Python program to to count the number of even and odd numbers from a series of numbers.

Sample numbers : numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]

Expected Output :

Number of even numbers : 5

Number of odd numbers : 4

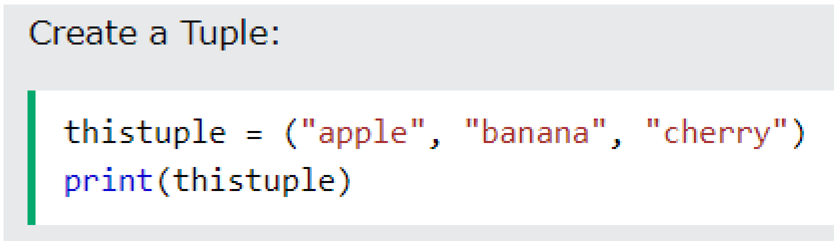
Exercise 5

Write a Python program to calculate the following expression:

$$M = \frac{1}{2} + \frac{2}{3} + \frac{3}{4} + \cdots + \frac{99}{100}$$

6. Python Tuples

- Tuples are used to store multiple items in a single variable.
- A tuple is a collection which is ordered and **unchangeable**.
- Tuples are written with round brackets.



Create a Tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

https://www.w3schools.com/python/python_tuples.asp

7. Python NumPy

7.1 NumPy Introduction

In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

7.2 Installation of NumPy

If you have Python and PIP already installed on a system, install it using this command:

```
C:\Users\Your Name>pip install numpy
```

python -m pip install numpy

If these commands fails, then use a python distribution that already has NumPy installed like, Anaconda, Spyder etc.

NumPy is usually imported under the **np** alias. In Python alias are an alternate name for referring to the same thing. Create an alias with the as keyword while importing:

```
import numpy as np
```

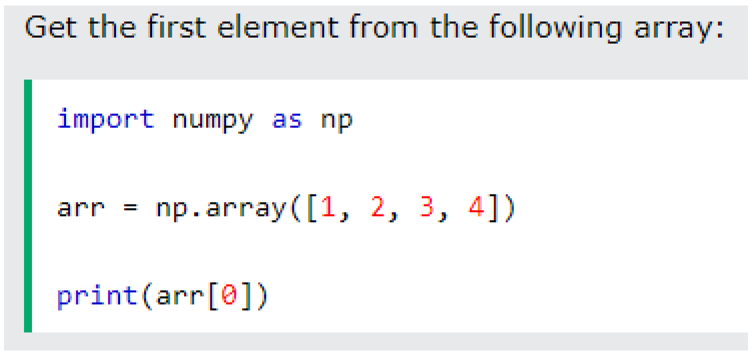
Now the NumPy package can be referred to as np instead of numpy.

7.3 Creating and accessing Numpy array

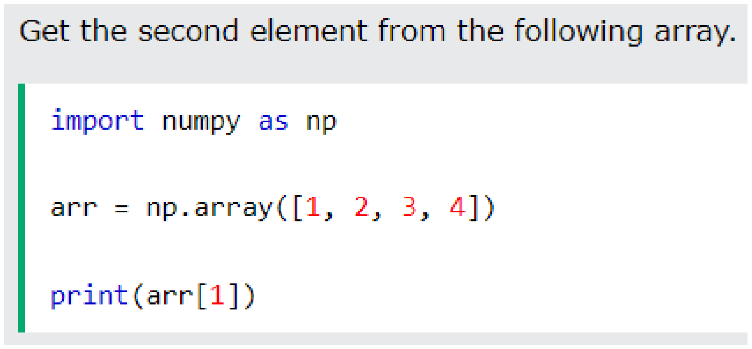
Create a NumPy ndarray object by using the **array**() function:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))
```

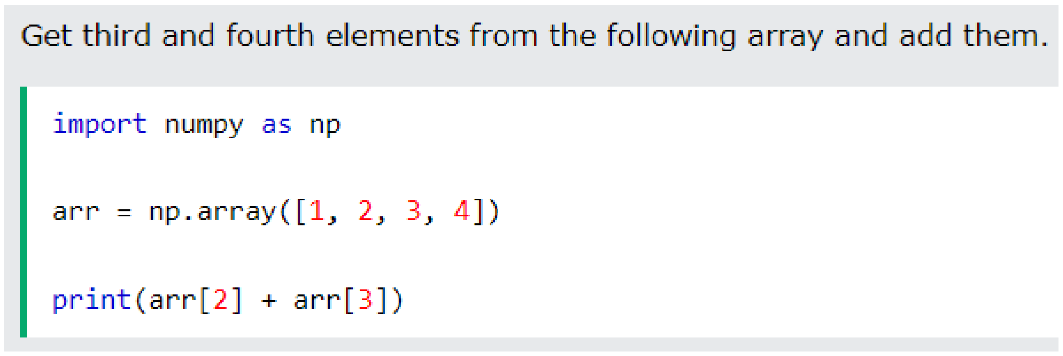
The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.:


Get the first element from the following array:

```python
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[0])
```


Get the second element from the following array.

```python
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[1])
```


Get third and fourth elements from the following array and add them.

```python
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[2] + arr[3])
```

7.4 Arithmetic Operators

You can perform arithmetic operations on these arrays. For example, if you add the arrays, the arithmetic operator will work element-wise. The output will be an array of the same dimension.

```python
import numpy as np

array1 = np.array([10,20,30,40,50])
array2 = np.arange(5)

print(array1 + array2)
print(array1 - array2)
print(array1 * array2)
print(array1*2)
print(array2**2)
```

Source: https://www.pluralsight.com/guides/overview-basic-numpy-operations

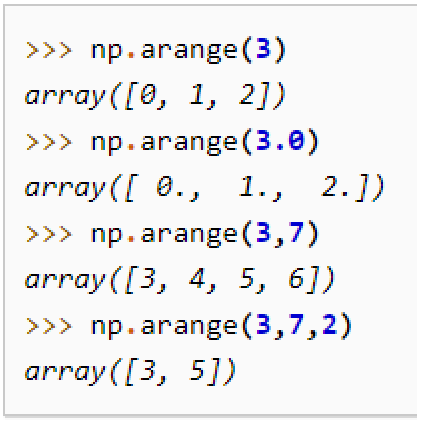
7.5 numpy.arange funtions

arange([start,] stop[, step,][, dtype]) : Returns an array with evenly spaced elements as per the interval. The interval mentioned is half-opened i.e. [Start, Stop)

Parameters:

- **start :** [optional] start of interval range. By default start = 0
- **stop :** end of interval range
- **step :** [optional] step size of interval. By default step size = 1. For any output *out*, this is the distance between two adjacent values, *out*[i+1] - *out*[i].
- **dtype** : type of output array

The advantage of numpy.**arange**() over the normal in-built **range**() function is that it allows us to generate sequences of numbers that are not integers.



```
>>> np.arange(3)
array([0, 1, 2])
>>> np.arange(3.0)
array([ 0.,  1.,  2.])
>>> np.arange(3,7)
array([3, 4, 5, 6])
>>> np.arange(3,7,2)
array([3, 5])
```

```
import numpy as np

# Printing all numbers from 1 to 2 in steps of 0.1
print(np.arange(1, 2, 0.1))
```

https://numpy.org/doc/stable/reference/generated/numpy.arange.html

```python
import numpy as np

#Print each even number in the array
arr = np.array([7, 3, 2, 10, 5, 0])
for x in arr:
  if x % 2 == 0:
    print(x)

#Print each even number between 0 and 10 (both included)
for x in np.arange(11):
  if x % 2 == 0:
    print("ex2", x)
```

For more usages, visit the website: https://www.w3schools.com/python/numpy/default.asp

Exercise 6
Write a NumPy program to create an array with integer values ranging from 12 to 38.

Exercise 7
Write a NumPy program to print common values between two arrays.

8. References

- Python Tutorial on the W3schools website: https://www.w3schools.com/python/default.asp
- Python Tutorial on the Tutorials Point website: https://www.tutorialspoint.com/python/index.htm

-- THE END --