CORVINUS UNIVERSITY of BUDAPEST

Python seminar 6.

Basic algorithms I.

Agenda

- Decision
- Summarization
- Counting
- Selection of an item with a given property
- Linear search
- Maximum selection

Decision

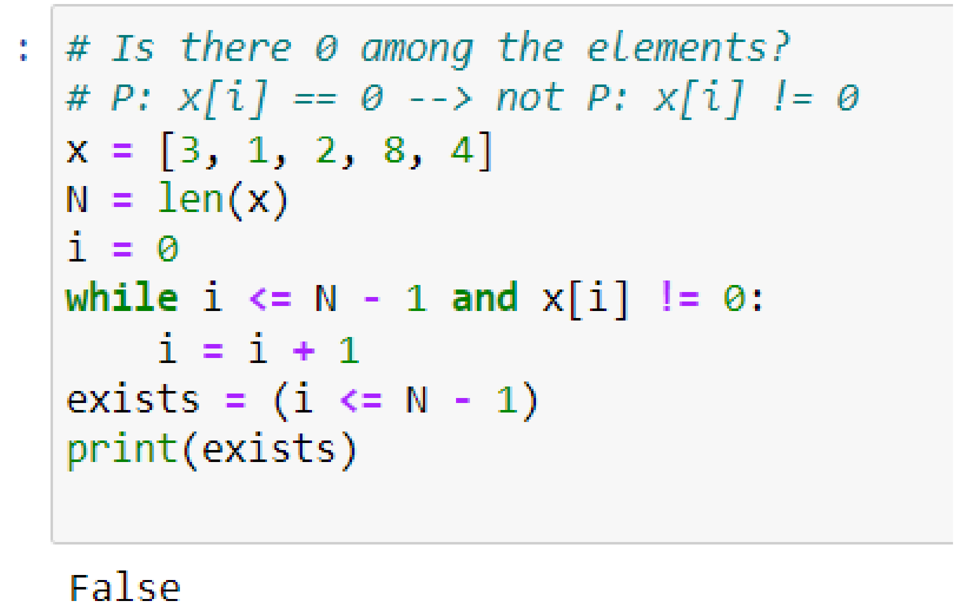
Is there an element with property P in the sequence?

N: number of elements, X: sequence, P: a given property
In Python, element numbering starts at 0, not 1 → i in 0..N-1

Pseudocode:

```
i = 1
loop while i <= N and ¬P(x[i])
    i = i + 1
end loop
exists = (i <= N)
```

Example



```python
# Is there 0 among the elements?
# P: x[i] == 0 --> not P: x[i] != 0
x = [3, 1, 2, 8, 4]
N = len(x)
i = 0
while i <= N - 1 and x[i] != 0:
    i = i + 1
exists = (i <= N - 1)
print(exists)
```

```
False
```

Summarization

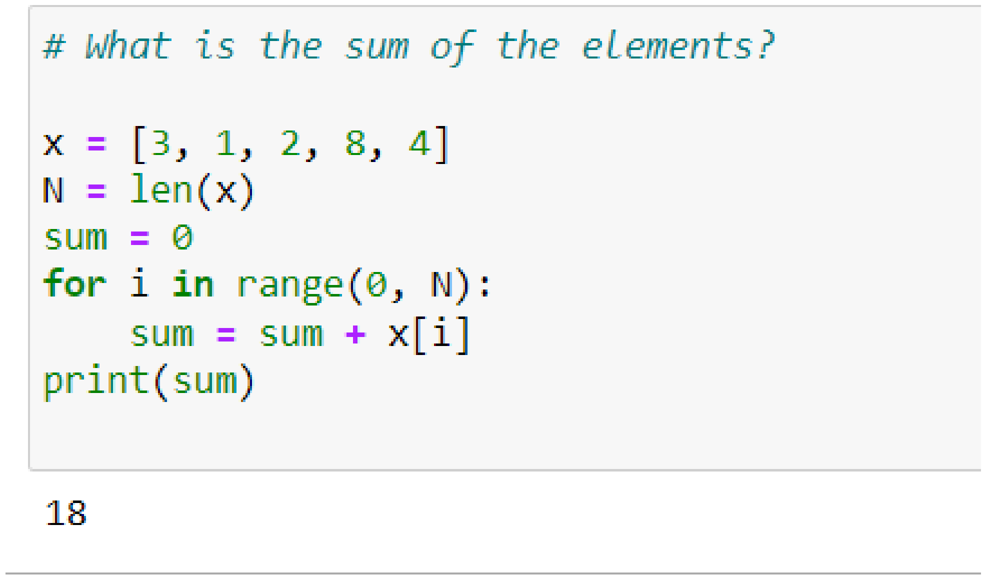
What is the sum of the elements of the sequence?

N: number of elements, X: sequence
In Python, element numbering starts at 0, not 1 → i in 0..N-1

Pseudocode:

```
sum = 0
loop for i in 1..N
    sum = sum + x[i]
end loop
```

Example



```python
# What is the sum of the elements?

x = [3, 1, 2, 8, 4]
N = len(x)
sum = 0
for i in range(0, N):
    sum = sum + x[i]
print(sum)
```

```
18
```

CORVINUS UNIVERSITY of BUDAPEST

Counting

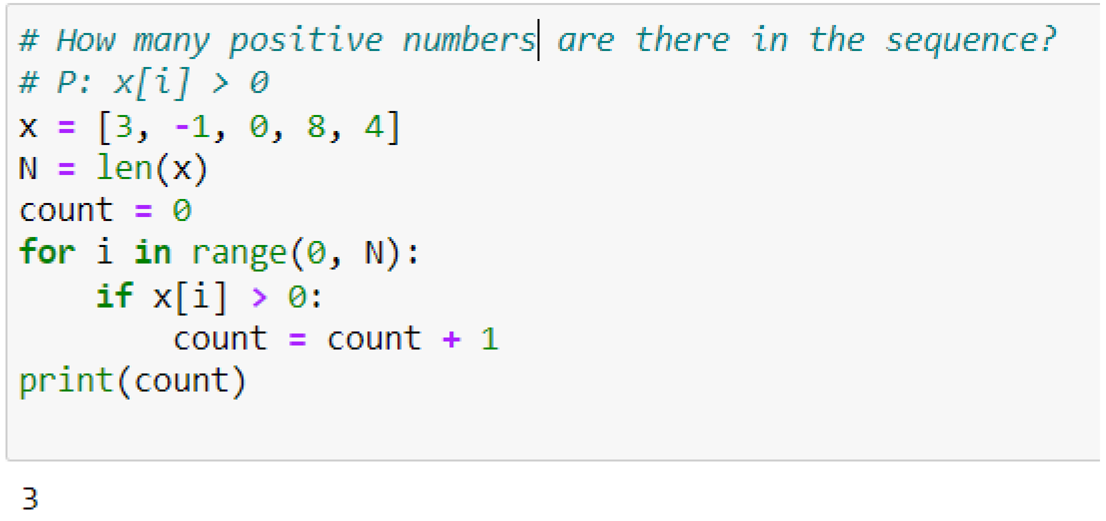
How many elements with property P are there in the sequence?

N: number of elements, X: sequence, P: a given property
In Python, element numbering starts at 0, not 1 → i in 0..N-1

Pseudocode:

```
count = 0
loop for i in 1..N
    if P(x[i]):
        count = count + 1
    end if
end loop
```

Example



```python
# How many positive numbers are there in the sequence?
# P: x[i] > 0
x = [3, -1, 0, 8, 4]
N = len(x)
count = 0
for i in range(0, N):
    if x[i] > 0:
        count = count + 1
print(count)
```

```
3
```

CORVINUS UNIVERSITY of BUDAPEST

Selection of an item with P

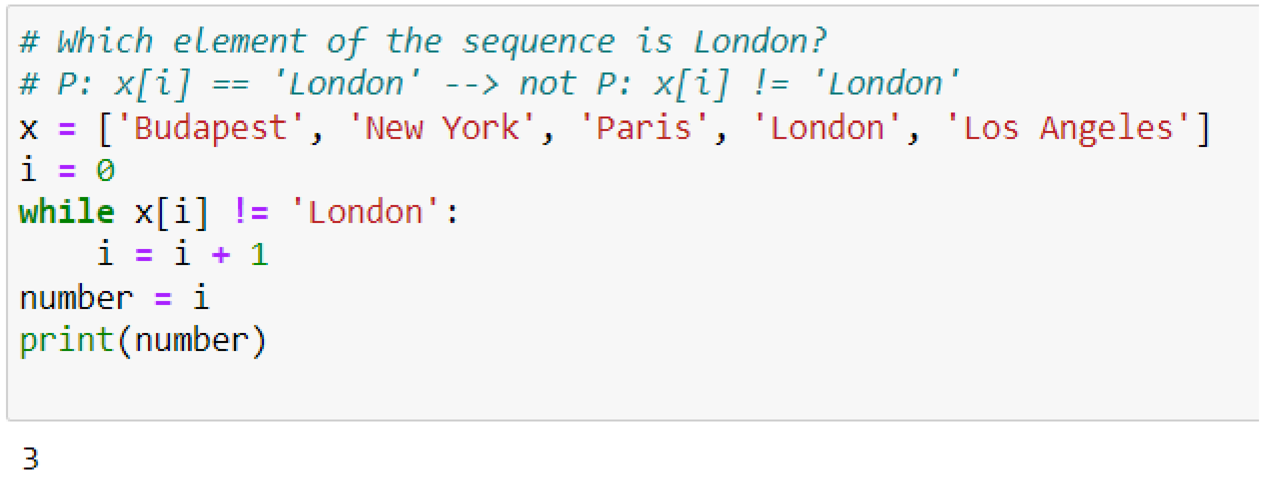
Which is the very first element with property P in the sequence? (assume there is one)

N: number of elements, X: sequence, P: a given property
In Python, element numbering starts at 0, not 1 → i in 0..N-1

Pseudocode:

```
i = 1
loop while ¬P(x[i])
    i = i + 1
end loop
number = i
# element = x[i]
```

Example



```python
# Which element of the sequence is London?
# P: x[i] == 'London' --> not P: x[i] != 'London'
x = ['Budapest', 'New York', 'Paris', 'London', 'Los Angeles']
i = 0
while x[i] != 'London':
    i = i + 1
number = i
print(number)
```

```
3
```

Linear search

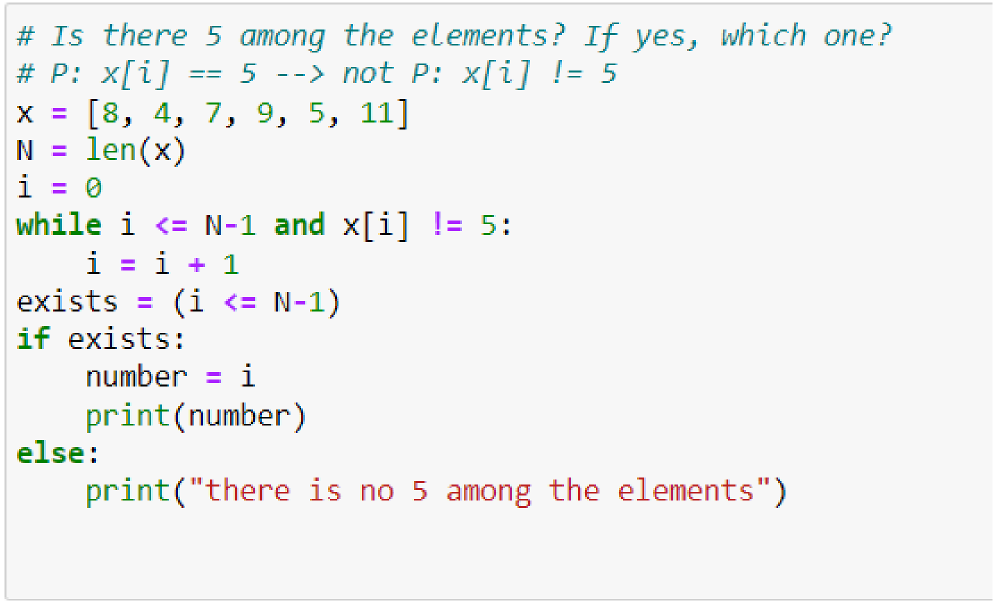
Is there an element with property P in the sequence?
If there is one, which one is the first?

N: number of elements, X: sequence, P: a given property
In Python, element numbering starts at 0, not 1 → i in 0..N-1

Pseudocode:

```
i = 1
loop while i <= N and $\neg$ P(x[i])
   i = i + 1
end loop
exists = (i <= N)
If exists
    number = i
End if
```

Example



```python
# Is there 5 among the elements? If yes, which one?
# P: x[i] == 5 --> not P: x[i] != 5
x = [8, 4, 7, 9, 5, 11]
N = len(x)
i = 0
while i <= N-1 and x[i] != 5:
    i = i + 1
exists = (i <= N-1)
if exists:
    number = i
    print(number)
else:
    print("there is no 5 among the elements")
```

```
4
```

Maximum selection

Which is the biggest item in the sequence?

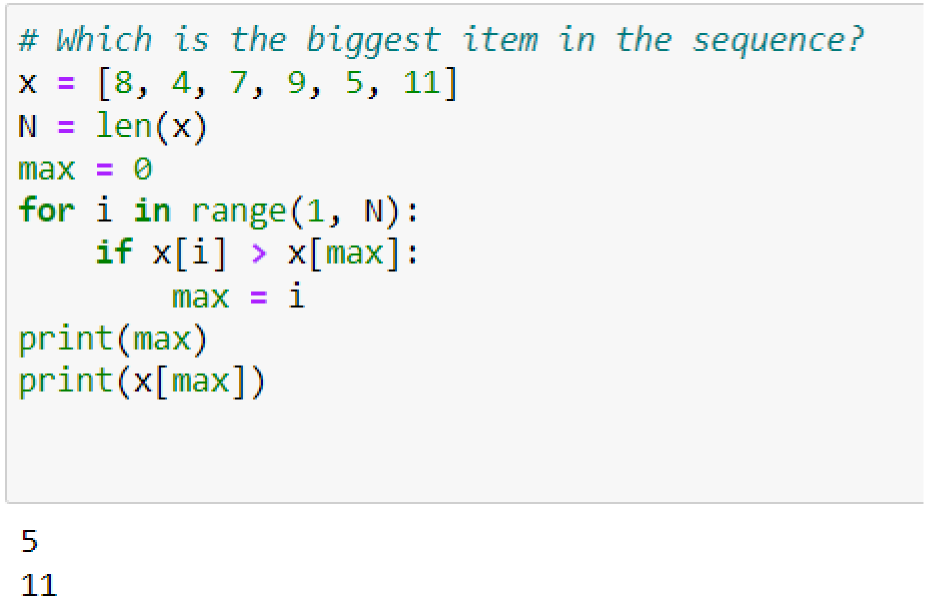
N: number of elements, X: sequence

In Python, element numbering starts at 0, not 1 → i in 0..N-1

Pseudocode:

```
max = 1
loop for i in 2..N
    if x[i] > x[max]
        max = i
end loop
```

Example



```python
# Which is the biggest item in the sequence?
x = [8, 4, 7, 9, 5, 11]
N = len(x)
max = 0
for i in range(1, N):
    if x[i] > x[max]:
        max = i
print(max)
print(x[max])
```

```
5
11
```

Python seminar 7.

Basic algorithms II.

CORVINUS UNIVERSITY of BUDAPEST

Agenda

- Copying
- Selection (elements with a given property)
- Separation
- Swap sort
- Minimum selection sort
- Bubble sort

CORVINUS UNIVERSITY of BUDAPEST

Copying

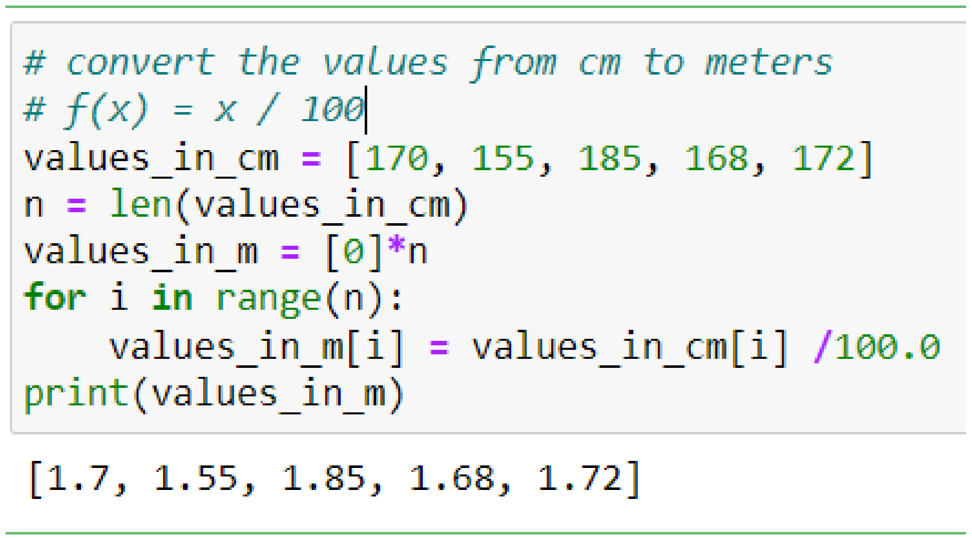
Let's transform the input sequence with element-by-element processing!

N: number of elements, X: input sequence, y: output sequence, f: function of transformation
In Python, element numbering starts at 0, not 1 ➔ i in 0..N-1

Pseudocode:

```
loop for i in 1..N
    y[i] = f(x[i])
end loop
```

Example



```python
# convert the values from cm to meters
# f(x) = x / 100
values_in_cm = [170, 155, 185, 168, 172]
n = len(values_in_cm)
values_in_m = [0]*n
for i in range(n):
    values_in_m[i] = values_in_cm[i] /100.0
print(values_in_m)
```

```
[1.7, 1.55, 1.85, 1.68, 1.72]
```

Selection (elements with P property)

Let’s select the elements of the input sequece with property P (or their indexes) into a new sequence!

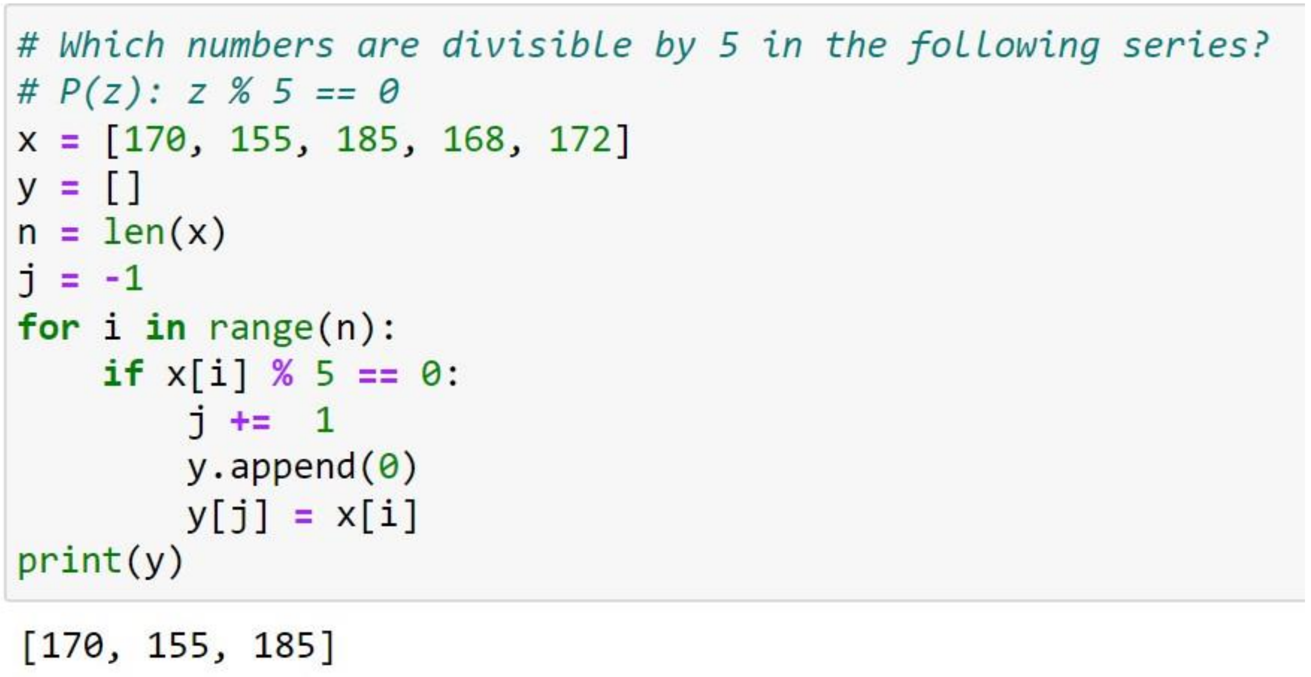
N: number of elements, X: input sequence, Y: output sequence, P: property

In Python, element numbering starts at 0, not 1 ➔ i in 0..N-1

Pseudocode:

```
j = 0
loop for i in 1..N
    if P(x[i])
        j = j + 1
        y[j] = x[i]       #or y[j] = i
    end if
end loop
```

Example



```python
# Which numbers are divisible by 5 in the following series?
# P(z): z % 5 == 0
x = [170, 155, 185, 168, 172]
y = []
n = len(x)
j = -1
for i in range(n):
    if x[i] % 5 == 0:
        j +=  1
        y.append(0)
        y[j] = x[i]
print(y)
```

```
[170, 155, 185]
```

Separation

Let’s separate the elements of a sequence into two new sequences depending on whether they have a property P or not.

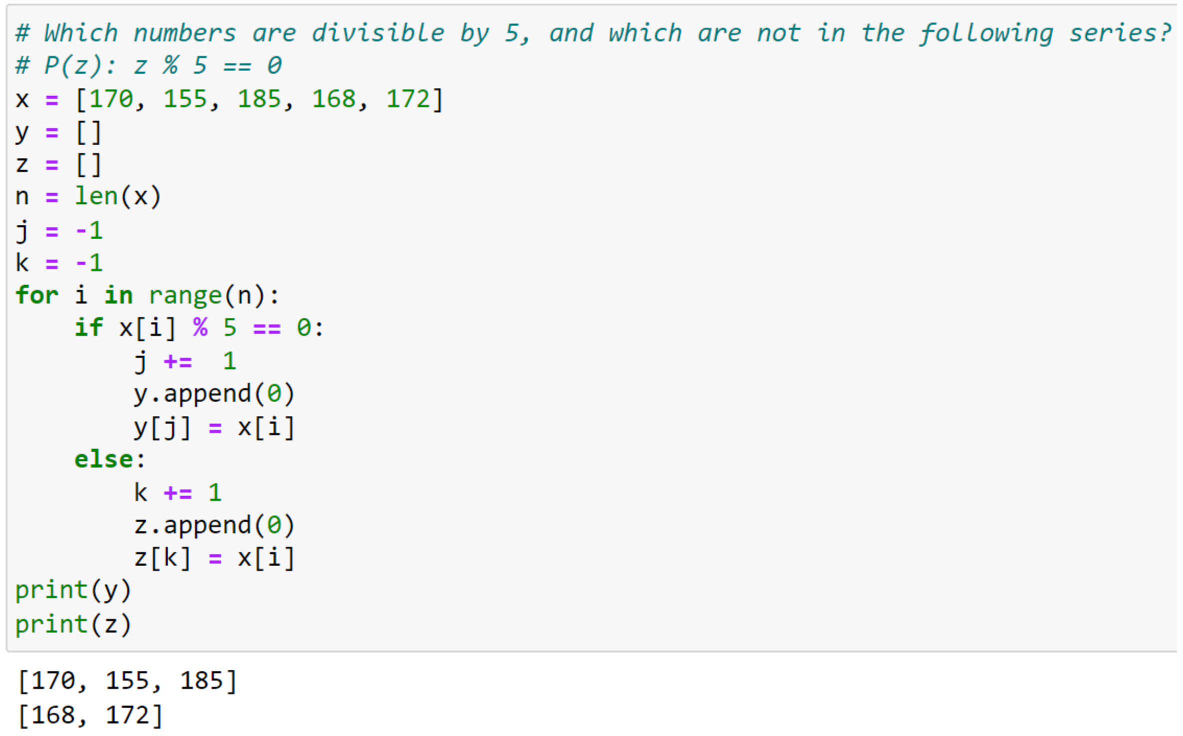
N: number of elements, X: input sequence, y, z: output sequences, P: a given property

In Python, element indexing starts at 0, not 1 ➔ i in 0..N-1

Pseudocode:

```
j = 0
k = 0
loop for i in 1..N
    if P(x[i])
        j = j + 1
        y[j] = x[i]        #or y[j] = i
    else
        k = k + 1
        z[k] = x[i]        #or z[k] = i
    end if
end loop
```

Example



```python
# Which numbers are divisible by 5, and which are not in the following series?
# P(z): z % 5 == 0
x = [170, 155, 185, 168, 172]
y = []
z = []
n = len(x)
j = -1
k = -1
for i in range(n):
    if x[i] % 5 == 0:
        j +=  1
        y.append(0)
        y[j] = x[i]
    else:
        k += 1
        z.append(0)
        z[k] = x[i]
print(y)
print(z)
```

```
[170, 155, 185]
[168, 172]
```

Swap sort

Go from left to right and compare each element with all the ones after it. If in one of the comparisons we find a smaller one than the given element, then let's swap the compared elements!

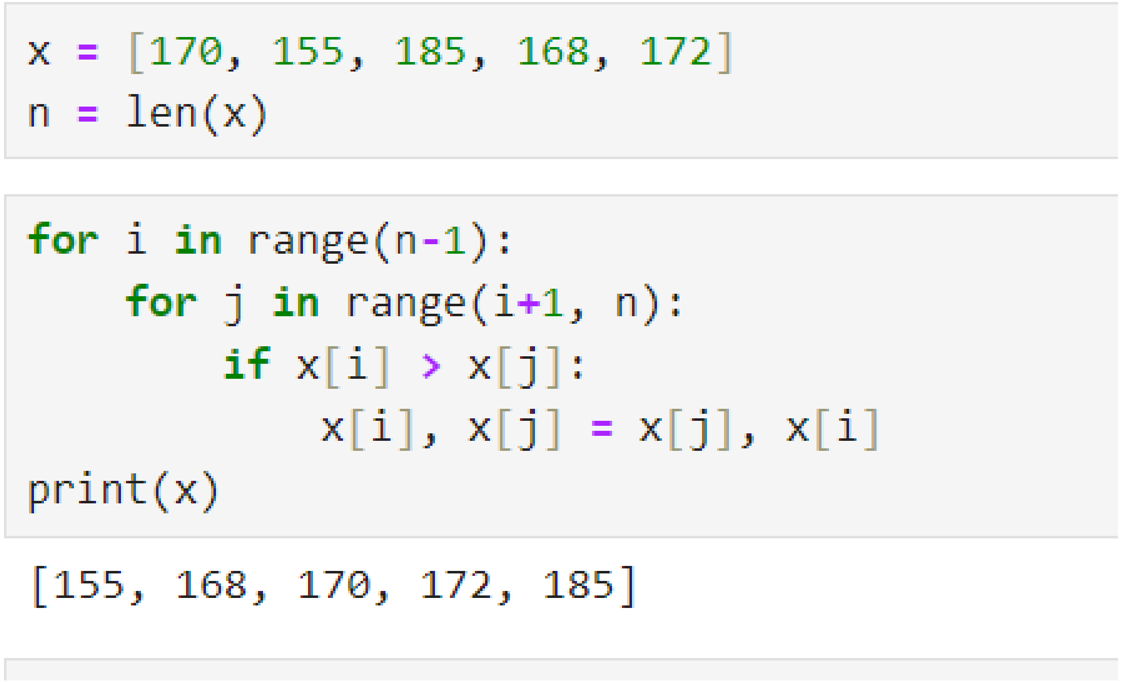
N: number of elements, X: sequence

In Python, element numbering starts at 0, not 1 ➔ i in 0..N-1

Pseudocode:

```
loop for i in 1..N-1
    loop for j in i+1..N
        if x[j] < x[i]
            swap(x[i], x[j])
        end if
    end loop
end loop
```

Example



```python
x = [170, 155, 185, 168, 172]
n = len(x)
```

```python
for i in range(n-1):
    for j in range(i+1, n):
        if x[i] > x[j]:
            x[i], x[j] = x[j], x[i]
print(x)
```

```
[155, 168, 170, 172, 185]
```

CORVINUS UNIVERSITY of BUDAPEST

Minimum selection sort

Go from left to right and compare each element with all the ones after it. Let’s swap the given element with the smallest one after it.

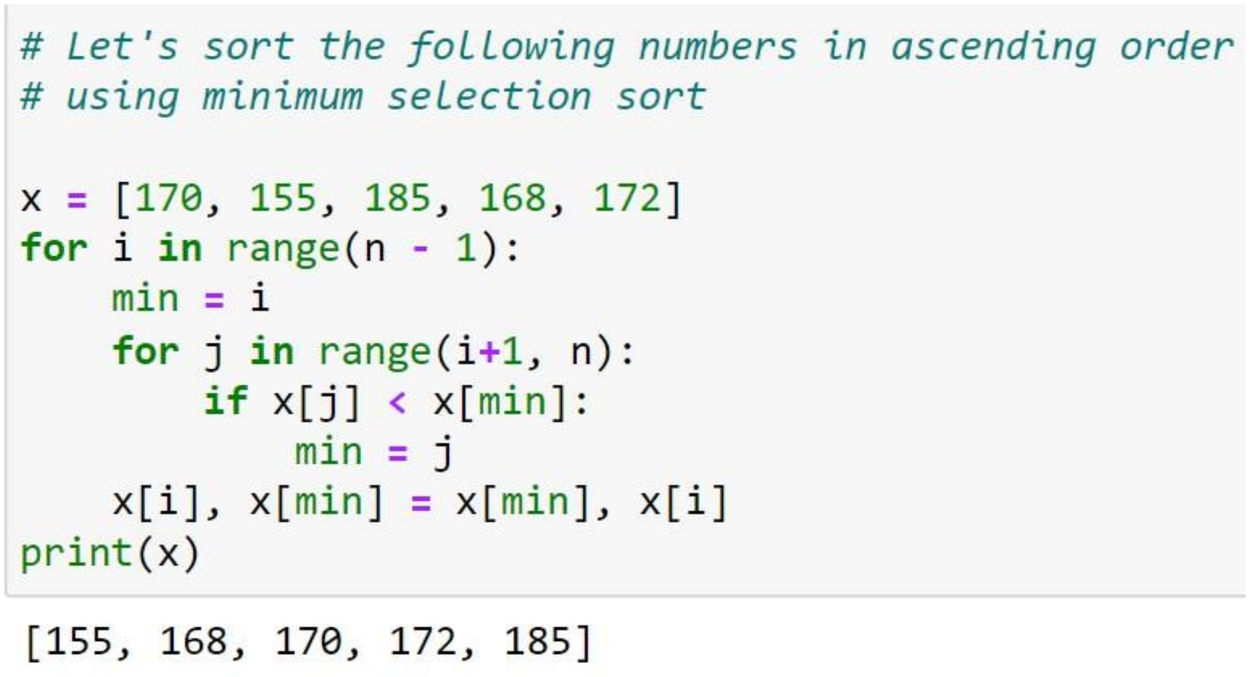
N: number of elements, X: sequence

In Python, element numbering starts at 0, not 1 ➔ i in 0..N-1

Pseudocode:

```
loop for i in 1..N-1
    min = i
    loop for j in i+1..N
        if x[j] < x[min]:
            min = j
    end loop
    swap(x[i], x[min])
end loop
```

Example



```python
# Let's sort the following numbers in ascending order
# using minimum selection sort

x = [170, 155, 185, 168, 172]
for i in range(n - 1):
    min = i
    for j in range(i+1, n):
        if x[j] < x[min]:
            min = j
    x[i], x[min] = x[min], x[i]
print(x)
```

```
[155, 168, 170, 172, 185]
```

CORVINUS UNIVERSITY of BUDAPEST

Bubble sort

Go from left to right and always compare adjacent elements. If the relationship between them is not appropriate, then we swap them. Let's repeat these steps so that we always skip the last element!

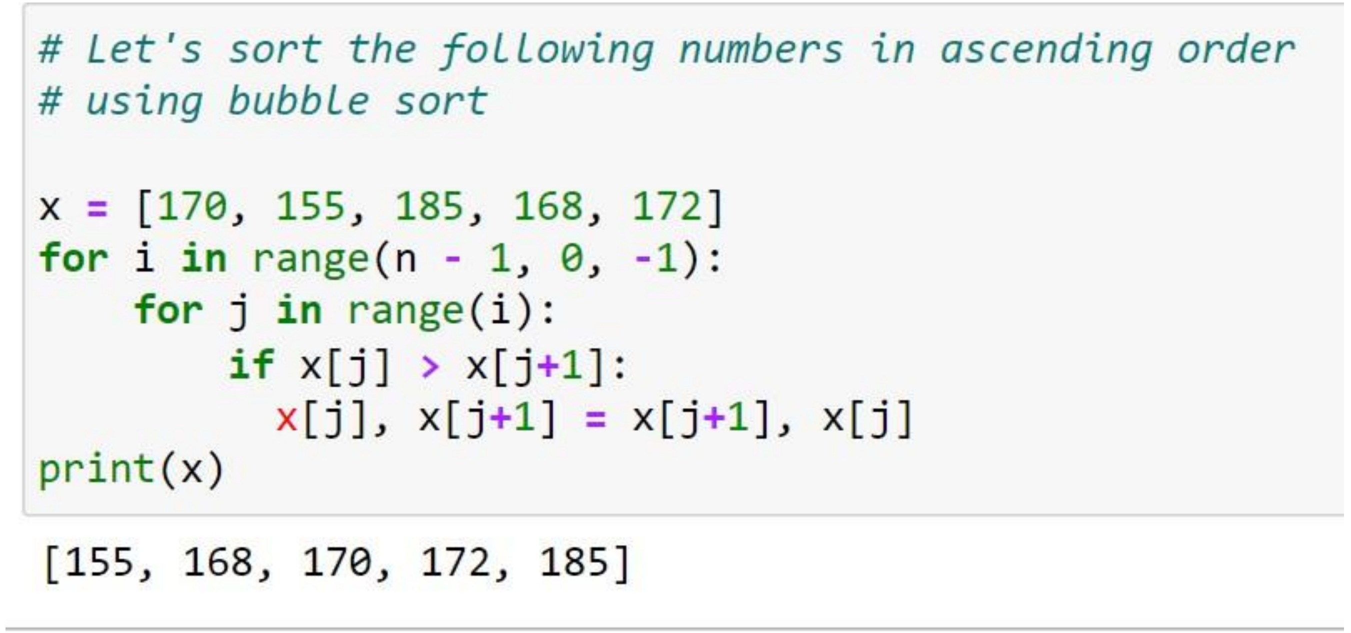
N: number of elements, X: sequence

In Python, element numbering starts at 0, not 1 ➔ i in 0..N-1

Pseudocode:

```
loop for i in N..2
    loop for j in 1..i-1
        if x[j] > x[j+1]:
            swap(x[j], x[j+1])
    end loop
end loop
```

Example



```python
# Let's sort the following numbers in ascending order
# using bubble sort

x = [170, 155, 185, 168, 172]
for i in range(n - 1, 0, -1):
    for j in range(i):
        if x[j] > x[j+1]:
            x[j], x[j+1] = x[j+1], x[j]
print(x)
```

```
[155, 168, 170, 172, 185]
```

CORVINUS UNIVERSITY of BUDAPEST

Python seminar 9.

Procedures and functions.
Modules.

Agenda

- User-defined functions and procedures in Python
- Passing parameters
- Recursion
- Local and global variables
- Lambda functions
- Modules and packages

User-defined functions and procedures*

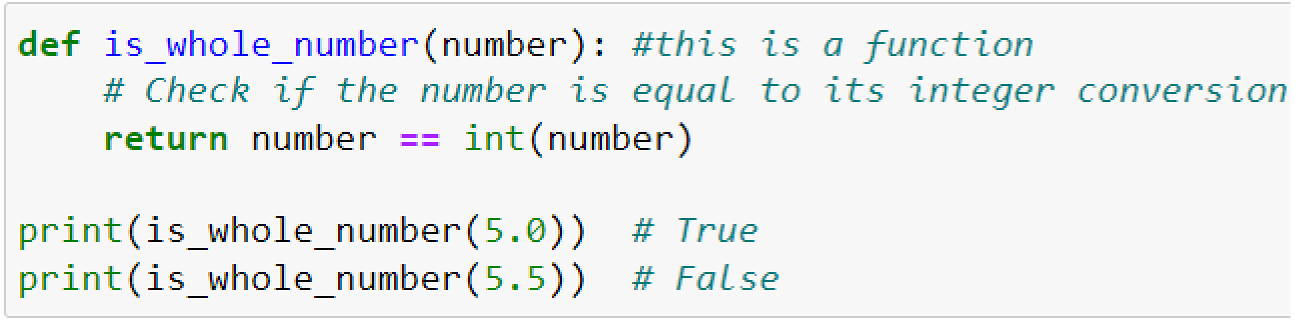
The (user-defined) function and the procedure are both a block of statements that perform a specific task. The function returns the result too, the procedure does not.

The syntax to declare a function or a procedure:

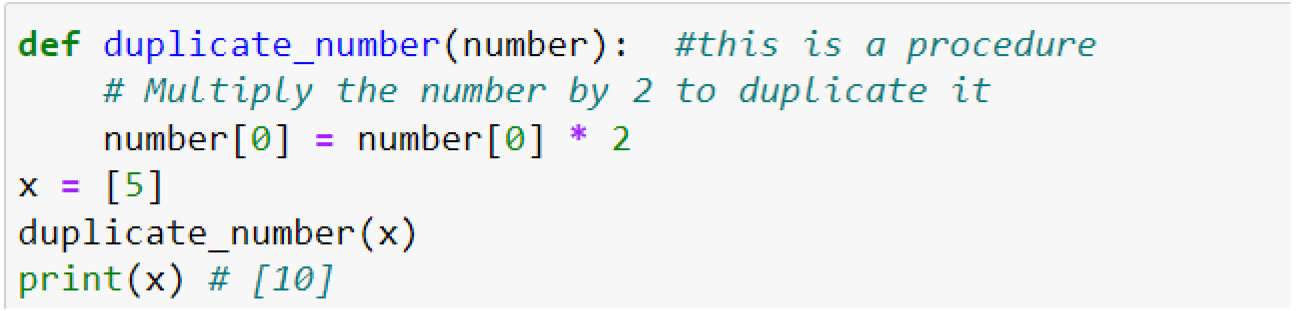
```
def  function_name(parameters):

    """ docstring """  #describes the function

    statements

    return expression
```

- At the procedures, the return statement is omitted
- The docstring is optional

Examples



```python
def is_whole_number(number): #this is a function
    # Check if the number is equal to its integer conversion
    return number == int(number)

print(is_whole_number(5.0))  # True
print(is_whole_number(5.5))  # False
```



```python
def duplicate_number(number):  #this is a procedure
    # Multiply the number by 2 to duplicate it
    number[0] = number[0] * 2
x = [5]
duplicate_number(x)
print(x) # [10]
```

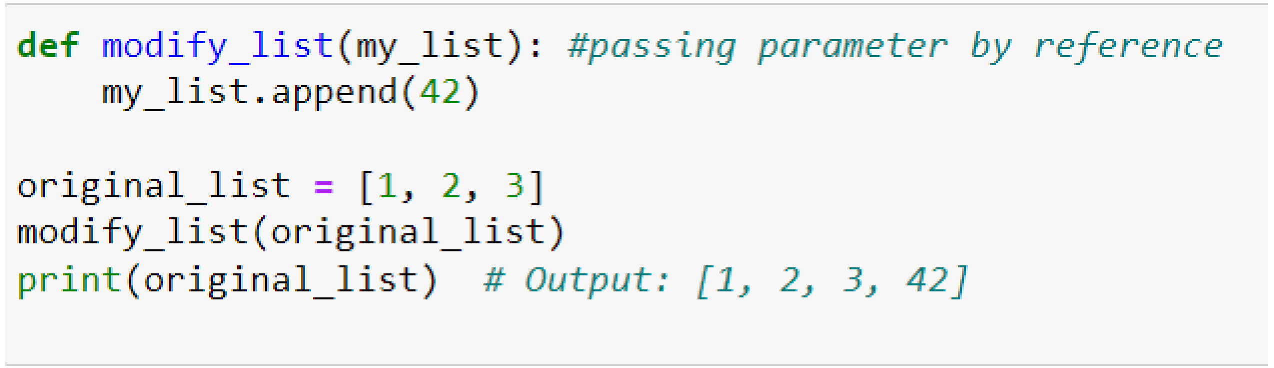
*Functions and procedure play an important role in code reusability, just as methods, modules, packages, libraries,   inheritance, decorators and compositions. (see some of them later)

Passing parameters

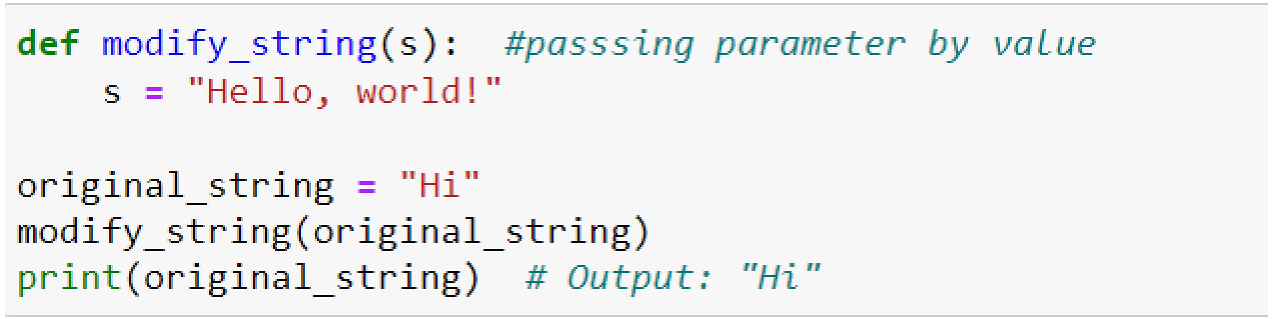
In Python, immutable data types (integers, floats, strings, tuples) are passed by value. In other cases, parameters are passed by reference.

- ❑ **Passing by value** means a copy of the original value. Any changes made to the parameter inside the function do not affect the original object outside the function.
- ❑ **Passing by reference** means that no copy of the original variable is created. So any changes made to the parameter inside the function will affect the original object outside the function.

Examples



```
def modify_list(my_list): #passing parameter by reference
    my_list.append(42)

original_list = [1, 2, 3]
modify_list(original_list)
print(original_list)  # Output: [1, 2, 3, 42]
```



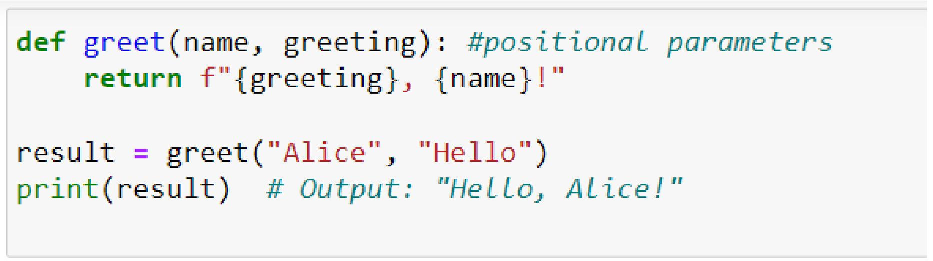
```
def modify_string(s):  #passsing parameter by value
    s = "Hello, world!"

original_string = "Hi"
modify_string(original_string)
print(original_string)  # Output: "Hi"
```

Parameter types

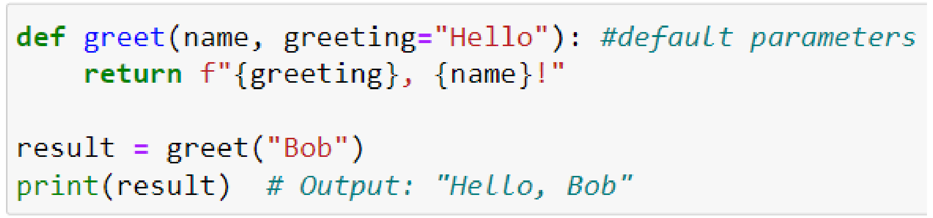
The types of parameters:

- ❑ **Positional parameters**: the order of the parameters is important
- ❑ **Default parameter**: assumes a default value if the value is not provided in the function call
- ❑ **Keyword parameters**: the parameters are specified with names, the order is not important
- ❑ **Arbitrary parameters**: can pass a variable number of parameters

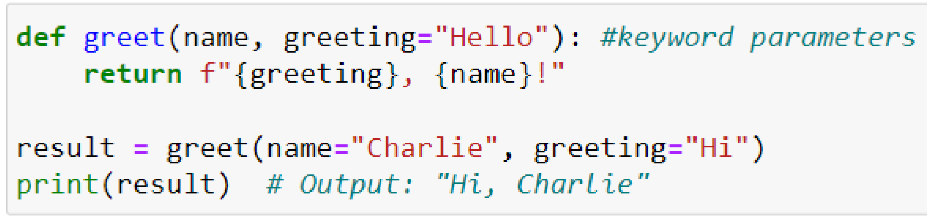
Examples



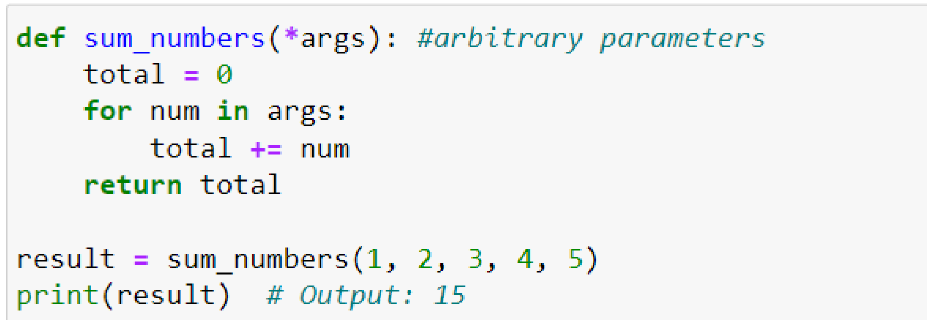
```python
def greet(name, greeting): #positional parameters
    return f"{greeting}, {name}!"

result = greet("Alice", "Hello")
print(result)  # Output: "Hello, Alice!"
```



```python
def greet(name, greeting="Hello"): #default parameters
    return f"{greeting}, {name}!"

result = greet("Bob")
print(result)  # Output: "Hello, Bob"
```



```python
def greet(name, greeting="Hello"): #keyword parameters
    return f"{greeting}, {name}!"

result = greet(name="Charlie", greeting="Hi")
print(result)  # Output: "Hi, Charlie"
```



```python
def sum_numbers(*args): #arbitrary parameters
    total = 0
    for num in args:
        total += num
    return total

result = sum_numbers(1, 2, 3, 4, 5)
print(result)  # Output: 15
```

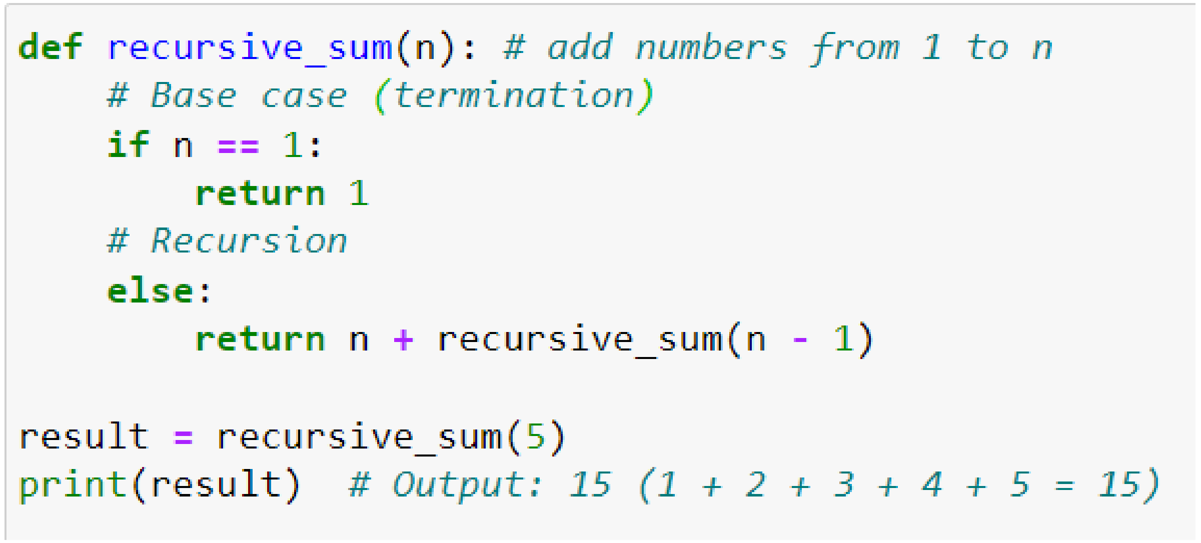
Recursion

Recursion refers to a technique where a function calls itself in order to solve a problem.

Instead of using loops to perform repetitive tasks …

- ❑ breaks down a problem into smaller, similar subproblems and solves them through successive calls to itself (recursion).
- ❑ This process continues until a base case is reached, at which point the function returns a result (termination).

Example



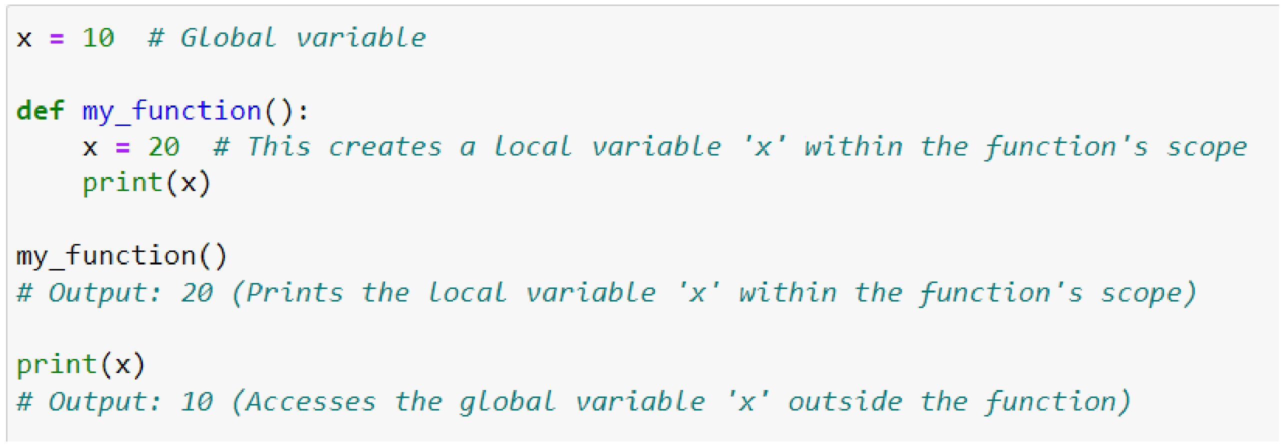
```python
def recursive_sum(n): # add numbers from 1 to n
    # Base case (termination)
    if n == 1:
        return 1
    # Recursion
    else:
        return n + recursive_sum(n - 1)

result = recursive_sum(5)
print(result)  # Output: 15 (1 + 2 + 3 + 4 + 5 = 15)
```

Local and global variables

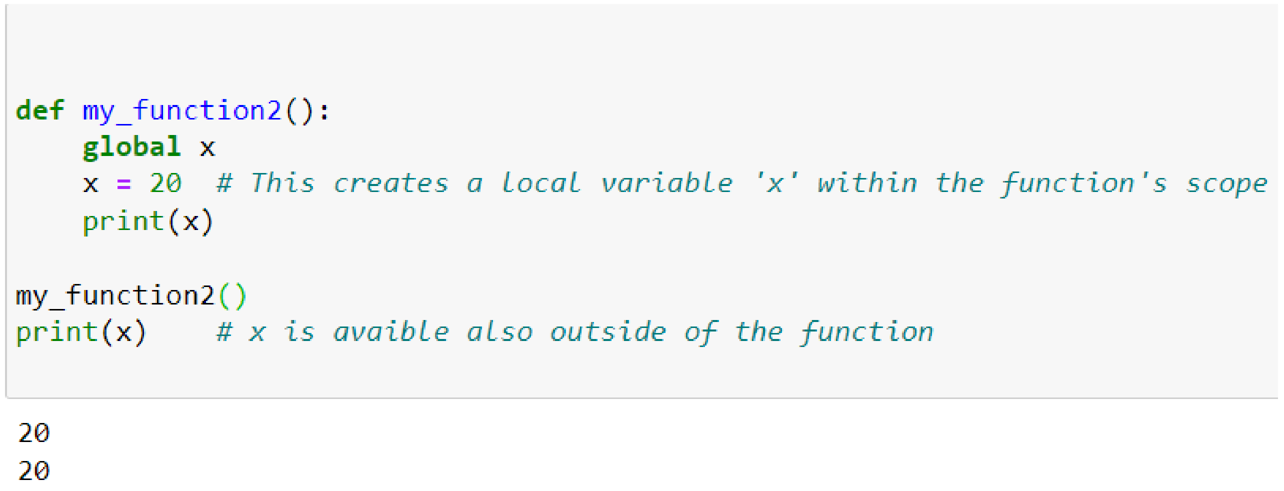
- **Local variables** are defined within a specific function or block of code, and they are only accessible and visible within that function or block.
- **Global variables** are defined outside of any function or block and can be accessed from anywhere in the code, both inside and outside functions.

The **global** keyword indicates, that we are working with the global variable rather than creating a new local variable.

Examples



```python
x = 10  # Global variable

def my_function():
    x = 20  # This creates a local variable 'x' within the function's scope
    print(x)

my_function()
# Output: 20 (Prints the local variable 'x' within the function's scope)

print(x)
# Output: 10 (Accesses the global variable 'x' outside the function)
```



```python
def my_function2():
    global x
    x = 20  # This creates a local variable 'x' within the function's scope
    print(x)

my_function2()
print(x)    # x is avaible also outside of the function
```

```
20
20
```

Lambda functions

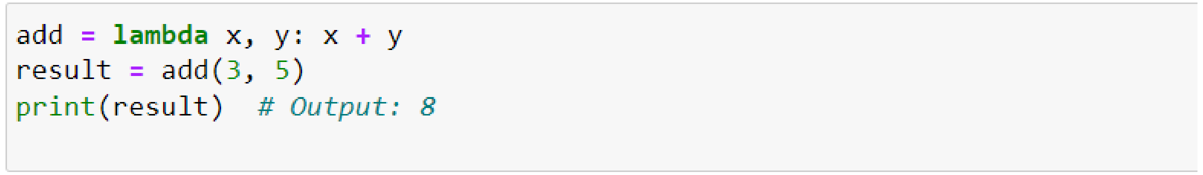
A lambda function is a small, anonymous, one-line function that can have any number of arguments (parameters) but can only have one expression.

The syntax to declare a lambda function:

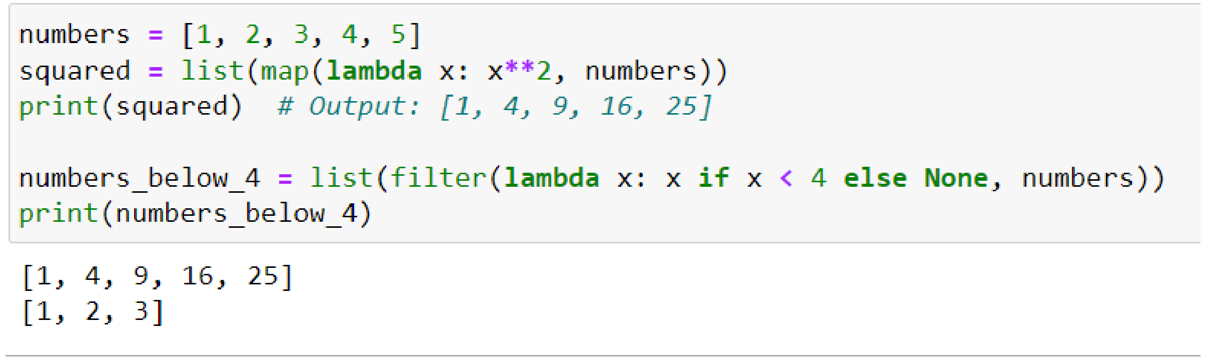
lambda arguments: expression

Lambda function is often combined with the …

- map() function to apply the same operation to every item of an iterable
- filter() function to select items from an iterable
- reduce() function to perform cumulative operations on an iterable

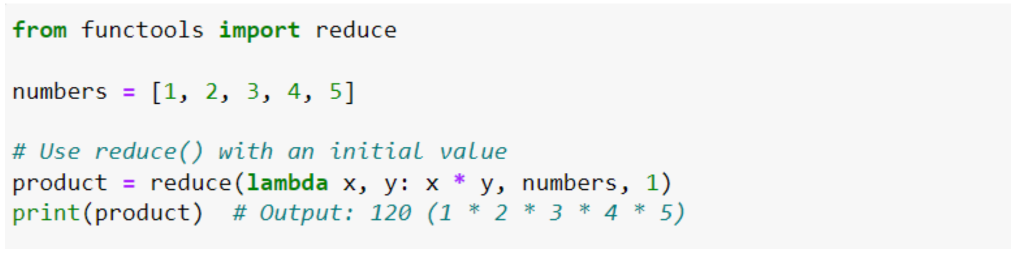


```python
add = lambda x, y: x + y
result = add(3, 5)
print(result)  # Output: 8
```



```python
numbers = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x**2, numbers))
print(squared)  # Output: [1, 4, 9, 16, 25]

numbers_below_4 = list(filter(lambda x: x if x < 4 else None, numbers))
print(numbers_below_4)
```

```
[1, 4, 9, 16, 25]
[1, 2, 3]
```



```python
from functools import reduce

numbers = [1, 2, 3, 4, 5]

# Use reduce() with an initial value
product = reduce(lambda x, y: x * y, numbers, 1)
print(product)  # Output: 120 (1 * 2 * 3 * 4 * 5)
```

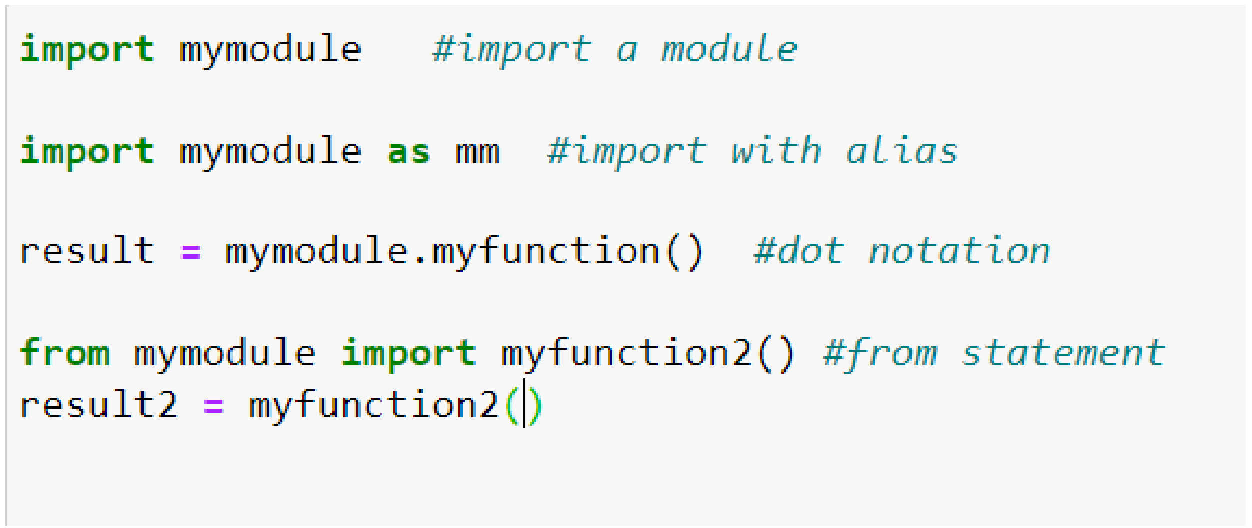
Lambda functions are used to solve only simple, short tasks.

Modules and packages

A module is a file containing Python code, including functions, classes, and variables, that can be used in other Python programs. Packages are directories containing one or more modules.

- ❑ **To create a module**, write the code in a .py file
- ❑ **To use the module**, you need to import it by the **import** modulname statement (without .py)
- ❑ You can **use an alias name** to make it easier the reference
- ❑ **You can access** the **modul elements** either by the **dot notation** or the **from** statement

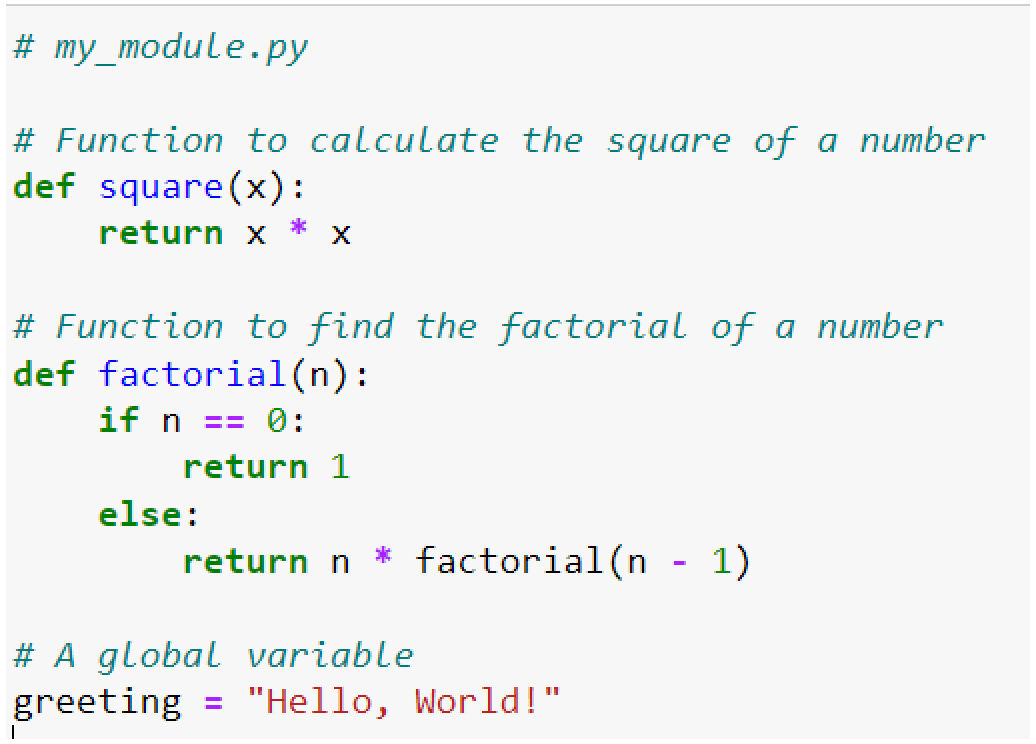
Examples



```
import mymodule    #import a module

import mymodule as mm   #import with alias

result = mymodule.myfunction()   #dot notation

from mymodule import myfunction2() #from statement
result2 = myfunction2()
```

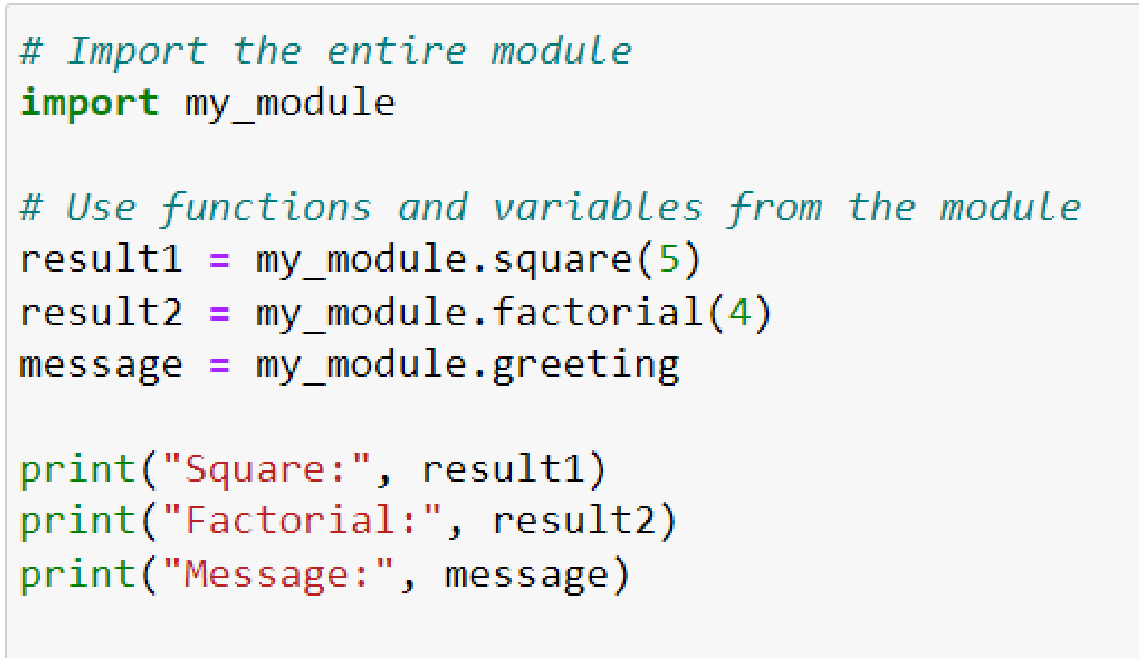
Module example

Module



```python
# my_module.py

# Function to calculate the square of a number
def square(x):
    return x * x

# Function to find the factorial of a number
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

# A global variable
greeting = "Hello, World!"
```

Module usage

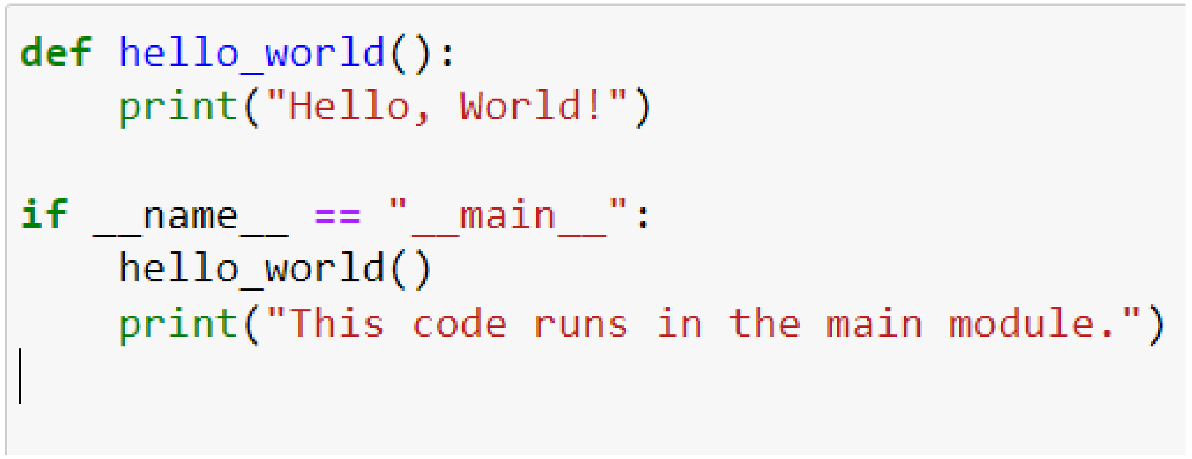


```python
# Import the entire module
import my_module

# Use functions and variables from the module
result1 = my_module.square(5)
result2 = my_module.factorial(4)
message = my_module.greeting

print("Square:", result1)
print("Factorial:", result2)
print("Message:", message)
```

The main module

When a Python script is executed, it is considered the "main" module.

- ❑ It is the starting point of the execution
- ❑ if __name__ == "__main__": allows you to specify code that should only run when the script is executed as the main module.
- ❑ It prevents code from running when the module is imported as a module in other scripts.



```python
def hello_world():
    print("Hello, World!")

if __name__ == "__main__":
    hello_world()
    print("This code runs in the main module.")
```

In Jupyter Notebook, the concept of a "main module" is not as relevant or significant as in traditional Python scripts or modules.

Python seminar 10.

Compound data types II.
Text files.

CORVINUS UNIVERSITY of BUDAPEST

Agenda

- Files in Python
- Basic text file operations
- File pointer positioning
- Create, rename and delete a text file
- Useful string methods for handling text files
- Handling exceptions

Files in Python

A file is a structured collection of data that are stored on the disk.

The file’s data structure can be various:

- **Sequential**. Data is stored in a linear manner from the beginning to the end. This is common in text files, where lines of text are stored one after another.
- **Hierarchical**. Data is organized in a tree-like fashion with parent-child relationships, such as JSON and XML
- **Tabular**. Dat is organized in a tabular structure, with rows and columns. Each line in the file corresponds to a record, and values are separated by commas (or other delimiters).
- **Binary**. Binary files contain raw binary data organized in a format specific to the application. For example, image files, audio files, and executable files.
- **Custom**. Data organizing way is defined by the application

Text files

There are several concepts related to text file handling :

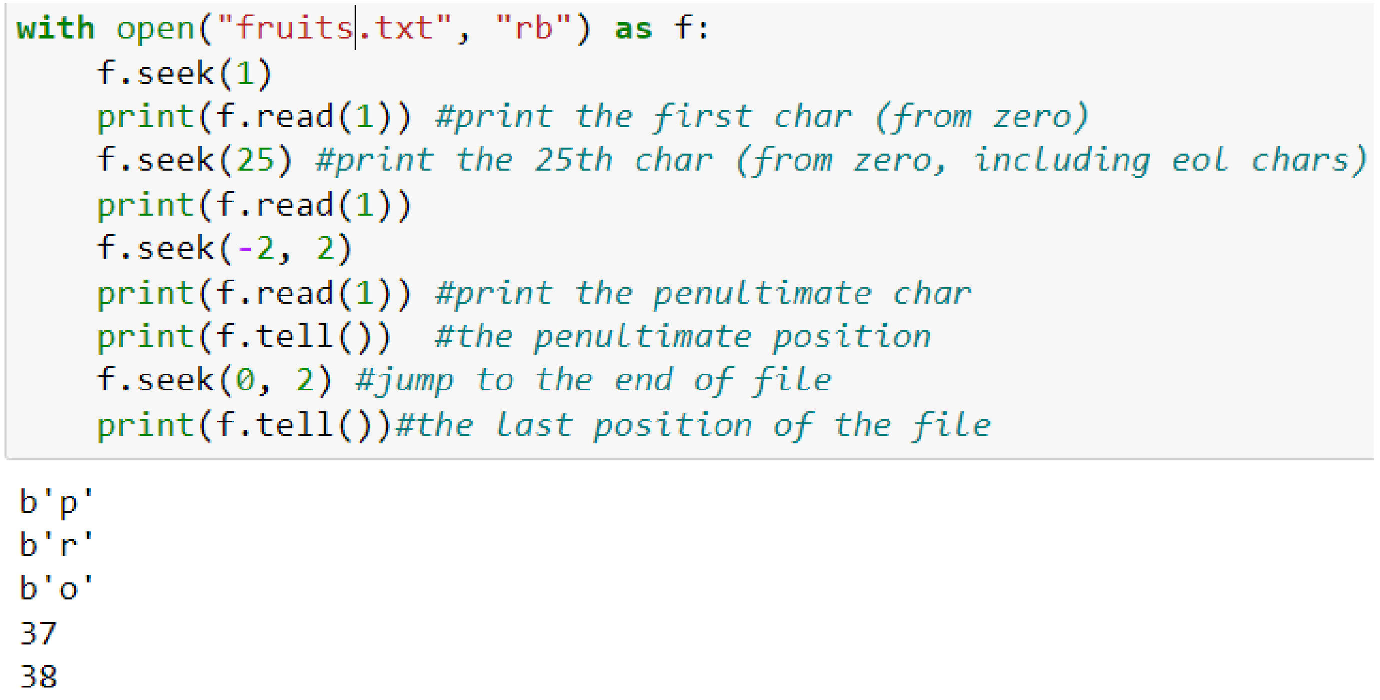
- **File Pointer.** The file pointer is a marker that represents the current position in the file. When you open a file, the file pointer is initially set to the beginning of the file. As you read or write data, the file pointer moves to keep track of the current position.
- **End-of-Line (EOL).** The end-of-line character represents the termination of a line in a text file. The specific character used as the end-of-line marker can vary between operating systems.
- **End-of-File (EOF).** The end-of-file marker indicates the end of the file. When reading a file, you can use the EOF marker to determine when you've reached the end of the file. In Python, this is typically detected by reaching an empty string when using read() or readline().
- **File Modes.** When opening a file, you specify a file mode that determines how the file should be opened (read, write, append, etc.). The default mode is the read.

File pointer positioning*

- **seek(offset,** whence) method changes the position of the file pointer**
  - **offset:** the number of bytes to move. A positive offset moves the pointer forward, a negative one moves it backward
  - whence: specifies the reference point.
    0 means the beginning of the file (default)
    1 means the current position
    2 means the end of the file
- **tell()** returns the current position of the file pointer

Examples

fruits.txt
apple orange
banana pear plum
melon



```
with open("fruits.txt", "rb") as f:
    f.seek(1)
    print(f.read(1)) #print the first char (from zero)
    f.seek(25) #print the 25th char (from zero, including eol chars)
    print(f.read(1))
    f.seek(-2, 2)
    print(f.read(1)) #print the penultimate char
    print(f.tell())  #the penultimate position
    f.seek(0, 2) #jump to the end of file
    print(f.tell())#the last position of the file
```

```
b'p'
b'r'
b'o'
37
38
```

*   The seek method should be used with caution in write mode
** To move backward, the file must be opened in binary mode, i.e. using rb as opening mode.

Basic text file operations and methods

Operation	Parameters	Description	Example

open()	**filename**
	mode*
	
	encoding
		Open a text file with a given filename
		Open a text file with a given mode
		r: only reading, w: write, a: append, r+: reading and writing existing file, w+: reading and writing, a+: append and read
		Open a text file with a given encoding
		UTF-8 (most commonly used)
			f = open("a.txt")
			f = open("a.txt", "r")
			
			f = open("a.txt", "r", "UTF-8")

close()	-	Closes the file	f.close()

read()
readline()
readlines()
	n
	n
	n
		Reads the whole file or reads n bytes from file into a string
		Reads one line or n bytes from file
		Reads each line or max. n bytes from file into a list
			s = f.read()
			line = f.readline()
			content = f.readlines()

write()
writelines()
	text
	list
		Writes the text into the file without adding any extra chars
		Writes the list into the file without adding any extra chars
			f.write(mystring)
			f.writeline(mylist)

append()	text	Appends the text to the file	f.append(mytext)

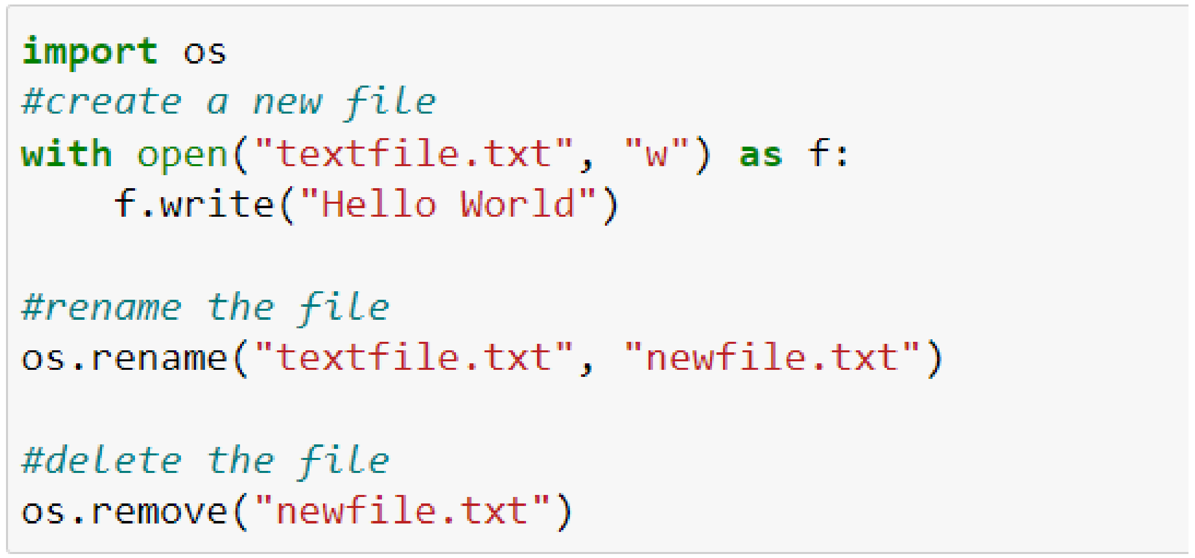
* At w, w+ modes the existing content will be overridden, at r+ mode can be overridden!

CORVINUS UNIVERSITY of BUDAPEST

Create, rename and delete a text file

- ❑ To create a new text file, you can use the **open()** function with mode 'w' (write). If the file already exists, it will be truncated; otherwise, a new file will be created.
- ❑ To rename a file, you can use the os.**rename()** function from the os module
- ❑ To delete (remove) a file, you can use the os.**remove()** function from the os module.

Examples

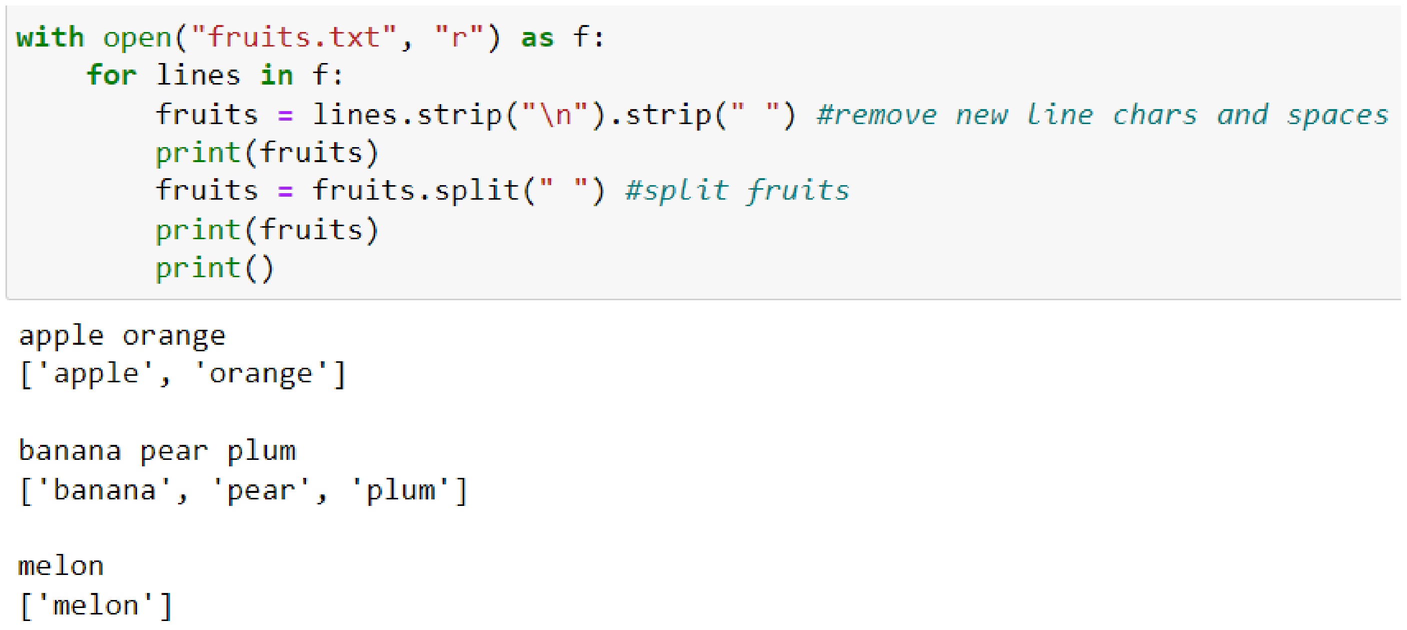


```python
import os
#create a new file
with open("textfile.txt", "w") as f:
    f.write("Hello World")

#rename the file
os.rename("textfile.txt", "newfile.txt")

#delete the file
os.remove("newfile.txt")
```

Useful string methods for text file handling*

- The **strip()** method in Python is a string method that is used to remove leading and trailing whitespaces (spaces, tabs, and newline characters) from a string.
- If you only want to strip from the left (leading) or right (trailing) side, you can use the **lstrip()** or **rstrip()** methods, respectively.
- The **split()** method in Python is a string method used to split a string into a list of substrings based on a specified delimiter.

Examples



```
with open("fruits.txt", "r") as f:
    for lines in f:
        fruits = lines.strip("\n").strip(" ") #remove new line chars and spaces
        print(fruits)
        fruits = fruits.split(" ") #split fruits
        print(fruits)
        print()
```

```
apple orange
['apple', 'orange']

banana pear plum
['banana', 'pear', 'plum']

melon
['melon']
```

* See also Seminar 5.

Exception handling

Exception handling is a programming construct that allows you to manage and respond to errors that may occur during the execution of a program.

The syntax to handling exception in Python:

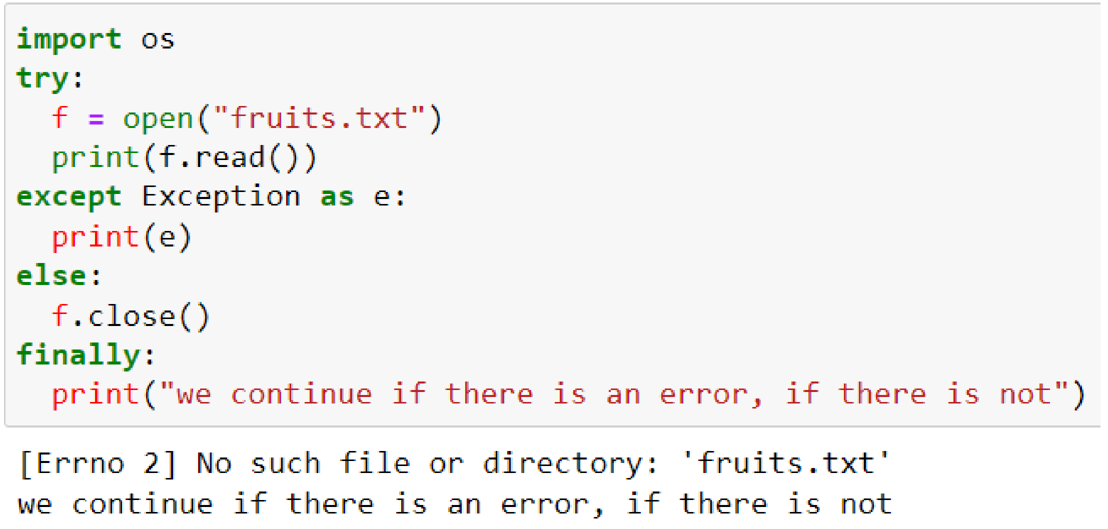
try:
block to contain the code that might raise an exception

except:
block to catch specific exceptions or handle general errors.

else:
block to execute code if no exceptions are raised in the try block. (Optional)

finally:
block to execute code that must run regardless of whether an exception occurred or not. (Optional)

Example:



```python
import os
try:
  f = open("fruits.txt")
  print(f.read())
except Exception as e:
  print(e)
else:
  f.close()
finally:
  print("we continue if there is an error, if there is not")
```

```
[Errno 2] No such file or directory: 'fruits.txt'
we continue if there is an error, if there is not
```