SU225 Python Basics

[Module 1 – Introduction 3](#_Toc53133138)

[Module 2 – Variables and types 4](#_Toc53133139)

[Module 3 – Flow constructs 8](#_Toc53133140)

[Module 4 – Files and I/O 10](#_Toc53133141)

[Module 5 – Object Oriented Programming 12](#_Toc53133142)

[Module 7 - Introduction to Multiprogramming 14](#_Toc53133143)

[Module 8 – Introduction to regular expressions 15](#_Toc53133144)

# Module 1 – Introduction

## Exercise 1: Locate the official Python documentation

1. Start your preferred browser and navigate to **python.org**
2. Go to the latest Python documentation
   1. Select **Documentation** / **Docs** from the menu
   2. Locate entry for the latest Python version (**Python x.x Resources**) and browse the documentation
   3. Enjoy this top level documentation page where you find:
      1. Language Reference -- Describes syntax and language elements
      2. Library Reference -- keep this under your pillow
      3. Glossary -- the most important terms explained
      4. Tutorial -- start here
      5. FAQs -- frequently asked questions (with answers!)
   4. Notice in the Other resources section to the left the PEP index link.
3. Find doc about **print()** in **Library Reference /** **Built-in Functions**. Notice all the other built-in functions.
   1. We will use many of these during the training.

## Exercise 2: Hello World

Hello Wold is a classic way to use a new language for the first time!

1. Start Your Python 3 Environment
   1. In this course we use **PyCharm**, which can be found in the start menu.
2. Create a new project: *New Project* or *File* / *New project*
   1. Use **PythonCourse** as name (last part of the path)
3. Rigth click the new project folder in on the left side of the UI
   1. Select *new* / *Python File*, name this **HelloWorld**
4. Enter code to print “Hello World” to the console

print( ‘Hello World’ )

1. Run the file
   1. Click the green arrow or press *Shift* + *F10*
2. The text are displayed in the Console window at the bottom of the UI

# Module 2 – Variables and types

## Exercise 1: The basics

1. Enter the following code in a new python script

print( "Apples: 5.00 kr.")

print( "Delivery charge: 32.50 kr." )

apples = int(input( "How many apples ? " ) )

total = apples \* 5

total = total + 32.50

print( f"To be payed in full: { total } kr." )

1. Enhance the program so that it is also possible to order pears (kr. 4.00), bananas (kr. 6.00) and pineapples (kr. 20.00).

## Exercise 2: Fruit loops and other healthy stuff!

1. Enter the program in a new python script:

01 goods = [ "Apples", "Pears", "Bananas", "Cherries" ]

02 prices = [ 5, 4, 6, 20 ]

03 count = []

04 sum = 0

05 for i in range( 0,3 ):

06 print( goods[i], " is ", prices[ i ], " kr." )

07 print( f"How many { goods[ i ] } do you want : " )

08 count[ i ] = input()

09 price = prices[ i ] \* count[i]

10 print( f"Cost: { price } kr.\n" )

11 sum = sum + price

12 price = 32.5

13 print( f"Delivery charge is { price} kr." )

14 sum = sum + price

15 print( f"Total : { sum } kr.\n")

Does it work? … and why the …. not ?

- the count list is empty – initialize it with 4 zeros (line 03)!

- input() returns a string: convert using int( … ) (line 08)

1. Add more fruits to the program.

## Exercise 3: Phone lists

1. Write a program, with a list of names and another of phone numbers.  
   Display the list of names (one at the time):
   * Olaf, Peter, Lars, John
2. The user must input the number (index) for the person.  
   The script must print the corresponding phone number (to be held in another list)

## Exercise 4: Spam, spam, spam and ... spam !

1. Using the following program as a template, create a script that reads the recipients name and substitutes “$XXX” with the entered name.  
   The result must be printed on screen

name = input( "Name of the victim ? " )

str = """

Most highly esteemed $XXX.

We have the incredible pleasure of introducing you to out new series of products for the discerning customer. As you, $XXX, is known as one who only goes for the best of the best, please allow us to offer you

a personal introductory discount.

In order for you to make use of this irresistable discount, $XXX, it is vital that you enter your full name, your visa account number and this code:

542XS3486Q-9, when you order from us.

We are looking forwards to making you, $XXX, a regular and esteemed customer.

Carl Smart.

CEO, Dan-junc.

"""

Use the call: str = str.replace( what , replacement )

*A template for the text is found in your home directory, subdirectory* ***spam.txt***

## Exercise 5: Age related problems

1. Write a simple program that asks for the users year of birth and prints out his age (Just hardcode the current year!)
2. Enter the following code in a new script file

agedict = { 'Ole' : 34, 'Peter' : 29, 'Lars' : 41 }

for name, age in agedict.items():

print( f'{ name } is { age } years old.' )

print()

agedict[ 'Ole' ] = 35

agedict[ "Hans" ] = 44

for name in agedict.keys():

print( f'{ name } is { agedict[ name ] } years old' )

Try to get the program to calculate the sum of all the age values.

# Module 3 – Flow constructs

## Exercise 1: The every popular guessing game !

1. Enter the following code in a new script file

import random

secret = int( random.random() \* 100 ) + 1

tries = 1

guess = int( input( "Guess a number between 1 and 100: " ) )

while secret != guess:

if guess > secret:

print( "Too high" )

else:

print( "Too low " )

guess = int( input( "Try again: " ) )

tries = tries + 1

print( "Correct! " )

print( f"You guessed it in { tries } tries" )

1. Enhance the program to say "Getting warmer", if the guess is closer than 10.

## Exercise 2: Simple math

1. Write a script that asks for ten numbers and calculates sum and average. It must also find the lowest and the highest number.  
   The result must be printed
2. Write a function that, given a number as parameter, can calculate sum and average for the numbers between one and the number.   
   Both sum and average must be returned from the function and printed

## Exercise 3: Odd temperatures

1. Write a script containing two functions, fahr() and celcius(), that can convert from Celcius to fahrenheit and from fahrenheit to celcius  
    T(°C) = (T(°F) - 32) × 5/9  
    T(°F) = T(°C) × 9/5 + 32
2. Test the functions with two numbers  
   Tip: 40°C 🡪 104°F

## Exercise 4: Strange strings

1. Write a function that takes a string as parameter.  
   Name the function Fst3
2. It must return the first three chars from the string, repeated three times. If there is less that three chars in the string, it must return the entire string
   * Fst3('Python') 🡪 PytPytPyt
   * Fst3('Na') 🡪 NaNaNa

## Exercise 5: To light or not to light !

1. Create a function that can help us decide whether it is time to turn on the bike lights.  
   Signature:

LightsOn(*biking*, *hour*)

*biking* (boolean) indicated whether we are biking at all

*hour* indicated the time of day.

1. Light must be on if the time is less than 8 or after 18 and we’re out biking.  
   Return True or False
2. Examples:
   1. LightsOn(False, 4) 🡪 False
   2. LightsOn(True, 5) 🡪 True
   3. LightsOn(True, 14) 🡪 False

# Module 4 – Files and I/O

## Exercise 1: Simple file I/O

1. Define a new script.  
   The script must ask for five names from the keyboard and write them directly to a new file.  
   Look at the resulting file in the file explorer ... have you forgotten something ?
2. Enhance the code.  
   Open the file for reading and read each name from the file and print it on the screen

## Exercise 2: Reading multiple files

In your home directory you will find the directory “su225”. This dir contains a subdirectory called “data”. In this dir you will find three text files, “Ane”, “Benny” and “Charlotte”, each containing some amusing nonsense poetry.

These files must be read and printed on the screen.

(Hint: a list of file-names might be a good approach !)

## Exercise 3: Average temperature on the surface of Mars (!)

In your home directory you will find the file “MPF\_subset\_data.csv”. This file origins from the actual Mars expedition and contains, one line at a time, the time and the surface temperature of Mars at a given time.

Select the relevant temperature (“column”) and calculate the average temperature.

Remember also to skip the first 2 lines (contains header text).

Open the file for reading and read each line into a variable. The variable must be split in order to gain access to time and temperature

arr = str.split( "," )

Since the content of the array is a string, each must be converted to floats in order for the average to be calculated. This can be done using:

temp = float( arr[ 4 ] )

Use the supplied data to calculated the average temperature and display this on screen. Expected result:

234 entries, averages: 1.0m: -49.491026, 0.5m: -48.793162, 0.25m: -48.342735

## Exercise 4: Handling spam and similar disgusting stuff

In your home directory you will find the file "syslog".  
This file contains a dump of a linux system log file that has been tampered with a bit: some lines contain the words "viagra.com" and "spam.com" (bummer !)

1. Open the file for reading. Read each line and check if the line contains one of the two trigger words.  
   If the line contains "viagra.com", add 1 to a variable viagra:

viagra = viagra + 1

If the line contains "spam.com", add 1 to a variable spam:

spam = spam + 1

Hint: to check use: *if “spam” in line:*

1. Finally, the program must display the number of occurrences of the two trigger words.

Note: You should actually also check (count), if data is accepted

# Module 5 – Object Oriented Programming

## Exercise 1: Build some Pyramids

In this context, a pyramid has two properties, side length and a height.

1. Define a new python class: Pyramid. To make it possible to construct pyramid objects, the class must have a constructor, This constructor must receive two floats

def \_\_init\_\_( self, side, height )

Add this constructor method to the class and test that you can create two objects:   
 p1: side: 10, height: 20  
 p2: side: 25, height: 80

1. Add a method called *volume* to calculate the volume of the pyramid based on the formula: Volume = side \* side \* height / 3.0

## Exercise 2: Specialized pyramids

A DensPyramid is a pyramid that contains a density (float) in addition to the normal side and height. Based on this information, it is possible to calculate the mass of the pyramid:

1. Define a new class **DensPyramid** inheriting from class **Pyramid**.  
   Add a constructor to receive three floats: side, height and density

def \_\_init\_\_( self, side, height, density )

1. Add a method “mass” to calculate the mass of a pyramid based on the formula: Mass = density \* volume  
   How would you go about using information already found in the standard pyramid?
2. Define two objects:  
    p1: side: 10, height: 20, density: 1.5  
    p2: side: 20, height: 80, density: 3.5  
   and print both their volume and their mass

## Exercise 3: Simple coordinates in a coordinate system

Simple coordinates contain two values – x and y.

1. Define a class coord that registers two values for x and y.

def \_\_init\_\_( self, x, y )

Verify that it is possible to define proper objects of the two classes

1. Define a method move(dx, dy), that will add dx to x and dy to y –

def move( self, dx, dy )

1. Add a method for adding coordinates. The method must return a coordinate  
   Note: This is actually an override for the **+** operator

def \_\_add\_\_( self, othercoord )

Use this template code

tmp = Coord( self.x, self.y )

tmp.x += othercoord.x

tmp.y += othercoord.y

return tmp

Verify the method by adding two coordinates.

1. Add a method for converting a coordinate into a string

def \_\_str\_\_( self )

This method must return a string formatted like “(2, -1)”

# Module 7 - Introduction to Multiprogramming

## Exercise 1: The basic thread class

1. Setup imports
   1. Thread from threading
   2. Time (to use when sleeping)
2. Design a thread class that subclasses (inherits from) the Thread class and takes two parameters in the constructor method, *name : str*  and *delay : int*.  
   These two values must be stored in the object
   1. Remember to call *\_\_init\_\_* on the super class!
3. Add a method run: **def run(self):**  
   This method will write the name on screen five times than wait for the indicated period of time (seconds), after each write.
4. Define a function: **main()**.  
   This function will create three thread objects having name and delay as parameter. It will then start each thread in sequence
5. Run the script a couple of times and look at the sequence of names being printed. This sequence might differ each time the script is run.  
   What conclusion may we draw from this ?

# Module 8 – Introduction to regular expressions

## Exercise 1: Search for a string

1. Start a new script.
2. Import the **re** module
3. Add a variable to hold a regular expression in the form

pattern = r' '

1. In between the ' ' add a regular expression to test.
2. Add a loop that inputs a sentence from keyboard, and tests whether the pattern is found.
   1. Print the result

## Exercise 2: Handling spam and similar disgusting stuff revisited

1. Reopen the project/script containing the spam filtering from exercise 4.4
2. An alternative to comparing strings, is to use a regular expression

import re

1. Change the code
   1. ***if spam in line ...***
2. to use the corresponding regular expression.
   1. Do not forget, that "." has a special meaning in RE, so the dot must be prefixed with a backslash

Did you get more or less matches?