

Course overview: Introduction to programming concepts

- What is a program?
- The Python programming language
- First steps in programming
- Program statements and data
- Designing programs.

This part will give an introduction to general programming concepts, and to the Python programming language.

Textbook and web

- We recommend a basic Python textbook such as “Python in easy steps” (Mike McGrath)
- Python can be downloaded for free from
[http : //python.org/downloads](http://python.org/downloads)
- In this course the computers already have Python installed.

What is a program?

- A program is a set of instructions which control a computer (laptop, desktop, tablet, etc)
- Programs can be written for huge variety of tasks: performing complex computations; financial trading; computer graphics and games; medical diagnosis and data processing; aircraft autopilot, etc
- Programs can read data from computer keyboard, mouse movements and actions, from data files, databases and any other sensors/data sources on the computer – and from internet if connected.
- Programs can present information graphically on computer screen, can write to text files, or update any other device connected to computer – if permitted to do so.

What is a program?

A set of instructions, in a particular order.

Example of simple program (expressed in English, not a programming language):

```
read a number X  
read a number Y  
calculate  $Z = (X + Y)$  divided by 2  
display Z
```

This computes average of two numbers, eg: for $X = 203$, $Y = 1965$, displays 1084.

What is a program?

- Another name for programs is *software* – as opposed to *hardware*, the physical computer and devices.
- Programs are written in text files in the format of some *programming language*
- The most widely-used programming languages are C, C++, C#, Java and Python.
- We'll use Python, as it's simplest of the popular languages.
- Python is intended to be simple language for learning programming – but can be used for real applications also.

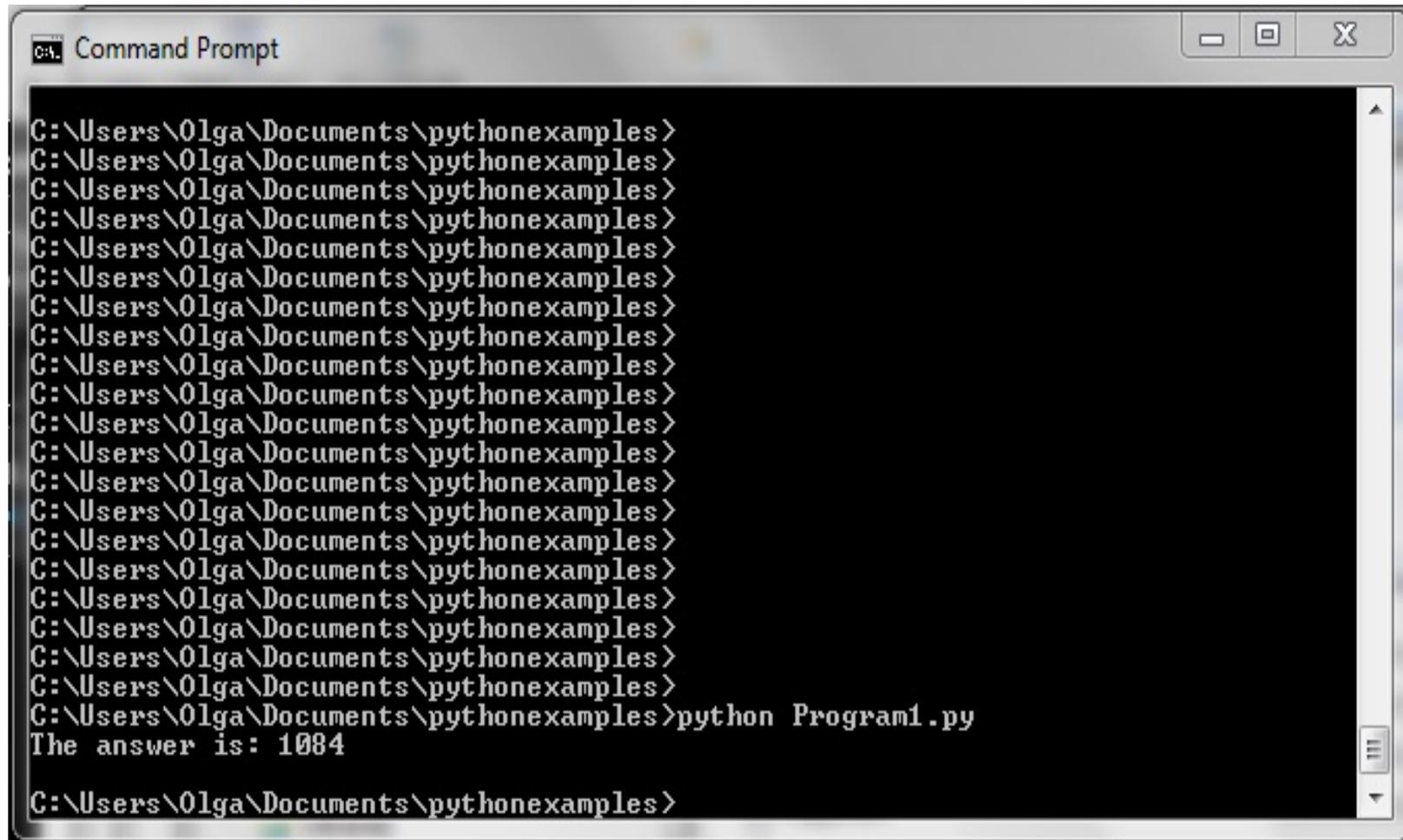
The Python programming language

- We write Python programs in text files (eg., using WordPad or Notepad), with .py file extension, eg.: *Program1.py*

```
X = 203
Y = 1965
Z = (X + Y)//2
print("The answer is: " + str(Z))
```

Here, three integer-valued variables X, Y, Z are declared. X and Y are given values 203 and 1965, then Z is calculated from them, and then displayed.

Python statements are written on successive text lines in text file *Program1.py*. `//` is used to divide integers. `str(Z)` converts number Z to a string.

A screenshot of a Windows Command Prompt window. The title bar reads "Command Prompt". The window contains a series of command-line prompts and outputs. The first 18 prompts are "C:\Users\Olga\Documents\pythonexamples>". The 19th prompt is "C:\Users\Olga\Documents\pythonexamples>python Program1.py". The output of this command is "The answer is: 1084". The final prompt is "C:\Users\Olga\Documents\pythonexamples>".

```
C:\Users\Olga\Documents\pythonexamples>
C:\Users\Olga\Documents\pythonexamples>python Program1.py
The answer is: 1084
C:\Users\Olga\Documents\pythonexamples>
```

Running Program1.py

First steps in programming

- Open the Windows console (Start; All Programs; Accessories; Command Prompt)
- In the Windows console, cd to the directory *cllexamples* where Program1.py is (on memory stick)
- Run the program with python:

```
python Program1.py
```

Note that *python* is lowercase *P* here.

X



Y



Z



**add, then divide
sum by 2**

Data in computer

First steps in programming

- A Python program in a file *Name.py* can have any number of statements, written on successive lines.
- *Give programs meaningful names:* *Average.py* would be better name for our first program.
- *Program statements:* individual instructions and steps the computer should take.

Eg.: `X = 203` “Introduce an integer variable called X, and assign the value 203 to it”.

`Z = (X + Y)//2` “Introduce an integer variable called Z, and assign $(X + Y)$ divided by 2 to it”.

Try changing the values assigned to X, Y and re-run using *python*.

First steps in programming

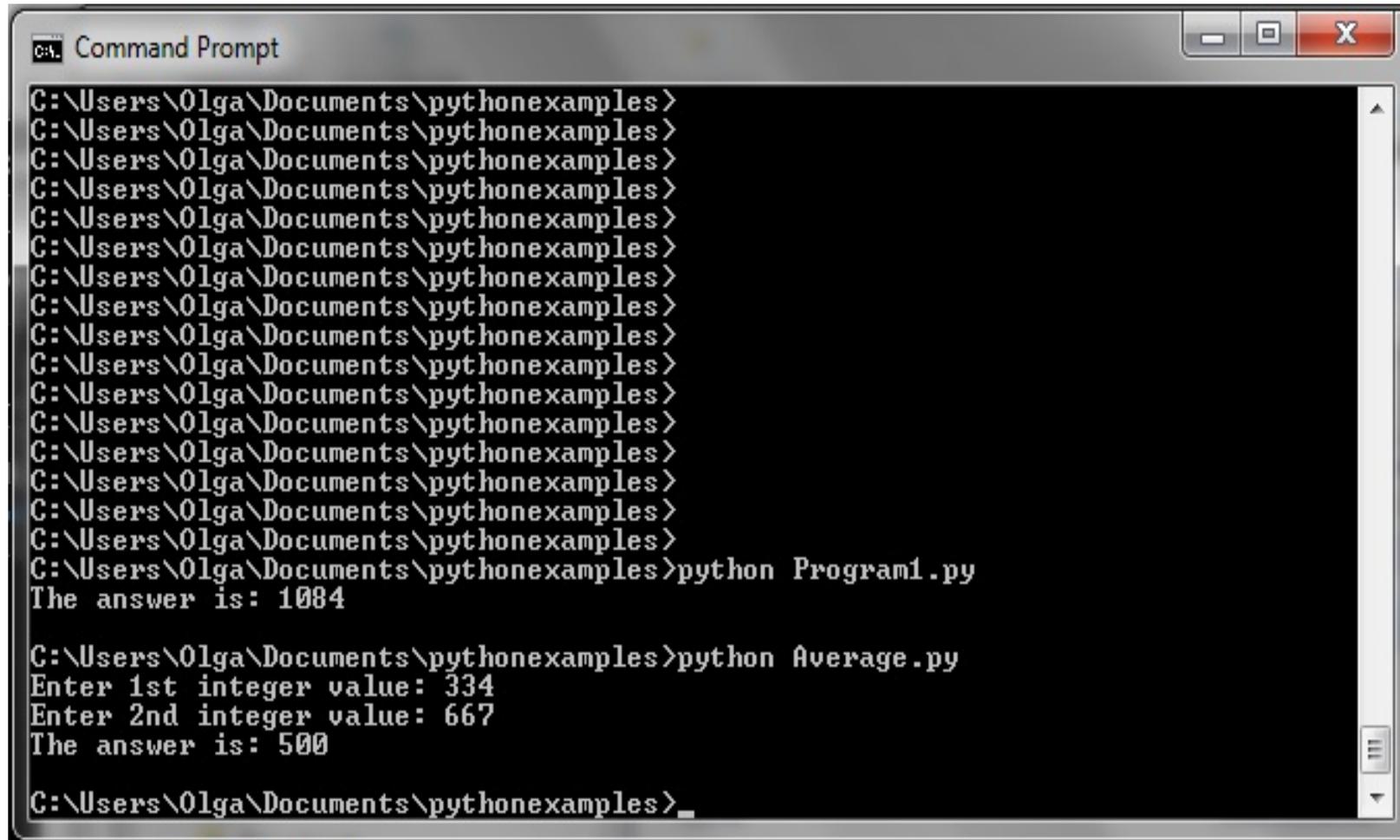
- Of course, a more useful program is one that can read inputs from user:

```
xvalue = input("Enter 1st integer value: ")
X = int(xvalue)
yvalue = input("Enter 2nd integer value: ")
Y = int(yvalue)
Z = (X + Y)//2
print("The answer is: " + str(Z))
```

The dialogs prompt user for the X, Y values.

Input from user is stored in String variables xvalue, yvalue – these store pieces of text.

Then converted to numbers by `int(value)`. Eg., string “334” is converted to number 334.



```
Command Prompt
C:\Users\Olga\Documents\pythonexamples>
C:\Users\Olga\Documents\pythonexamples>python Program1.py
The answer is: 1084

C:\Users\Olga\Documents\pythonexamples>python Average.py
Enter 1st integer value: 334
Enter 2nd integer value: 667
The answer is: 500

C:\Users\Olga\Documents\pythonexamples>_
```

Average.py input dialog

First steps in programming

- If we enter some wrong data (not integers), our program crashes – a professional program should be robust & continue despite errors
- The *User Interface* of a program consists of the visual components used to exchange data between program and users – in this case the input dialog and command window. Also called *Graphical User Interface* (GUI).

Program Statements

- Programs consist of *program statements* – individual processing instructions, operating on variables and values

- *Assignment statements* assign a value to a variable:

X = 530

These also declare/introduce the variable, so that it can be used in following statements:

X = 530

Y = X*X

- *Conditional statements* enable decisions to be made: if a condition is true, one behaviour is executed; if condition is false another behaviour is executed.

Max.py:

```
xvalue = input("Enter 1st integer value: ")
X = int(xvalue)
yvalue = input("Enter 2nd integer value: ")
Y = int(yvalue)
Z = 0
if X < Y :
    Z = Y
else :
    Z = X
print("The largest value is: " + str(Z))
```

The statement

```
if X < Y :  
    Z = Y  
else :  
    Z = X
```

tests the condition $X < Y$, ('is X less than Y?'), if this is true then the statement $Z = Y$ is executed, otherwise (if X equals Y or is larger than Y), the statement $Z = X$ is executed.

Effect is to always set Z to be the larger of X and Y.

Indent sub-statements such as $Z = X$ at least 2 spaces from their parent statement (here, the if-else statement).

Multiple decisions

Conditions can be as complex as needed, and multiple conditions can be tested (StudentMarks.py):

```
markvalue = input("Enter the student's mark: ")
mark = int(markvalue)
if mark < 40 :
    result = "Fail"
else :
    if mark >= 40 and mark < 50 :
        result = "Pass -- grade D"
    else :
        if mark >= 50 and mark < 60 :
            result = "Pass -- grade C"
        else :
            if mark >= 60 and mark < 70 :
                result = "Pass -- grade B"
```

```
    else :  
        result = "Pass -- grade A"  
print("The student's result is: " + result)
```

For each nested statement, indent a further 2 (or more) spaces.

The “and” keyword combines two conditions by conjunction.

result is a String variable – it stores a piece of text.

Conditional Statements

Examples of different cases:

<i>Input</i>	<i>Output</i>
10	“Fail”
42	“Pass – grade D”
55	“Pass – grade C”
67	“Pass – grade B”
70	“Pass – grade A”

Try executing StudentMarks.py with other example input values.

Loop Statements

- Loop statements repeat some instructions over & over again until a task is completed

- *for statements* loop for a fixed number of times:

```
for i in range(a,b+1) :  
    statements
```

means ‘execute the *statements* with $i = a$, then with $i = a+1$, $i = a+2$, ..., then with $i = b$ ’

- If $a > b$, does nothing
- If a equals b , just one iteration, for $i = a$.

Loop Statements

SumNumbers.py:

```
nvalue = input("Enter the number to sum: ")
n = int(nvalue)
sum = 0
for i in range(1,n+1) :
    sum = sum + i
print("The sum is: " + str(sum))
```

calculates sum of numbers from 1 to n.

Results for different input values:

n	sum
0	0
1	1
2	3
3	6
4	10
5	15
6	21
7	28
8	36
9	45
10	55

- Simple program structure as list of statements
- Variables of integer type (`X = 0`) – these can store values of integers, eg.: -214, 0, 1, 55, 1000, 500000, etc.
- Variables of String type (`s = ""`) – these can store strings/text, eg.: “Result”, “Enter a value”
- Assignment statements *var = value*
- Conditional statements

```
if Condition :
    statement1
else :
    statement2
```
- Bounded loops:

```
for var in range(a, b) :
    statement
```

More advanced Python

- `float` variables (rational numbers)
- Unbounded loops (`while` statements)
- What is programming?
- Functions and recursion
- List variables.

Calculations with rational numbers

- `float` variables can store fractional values: 0.5, 2.25, 0.0001, etc.
- Python provides functions for square root: `math.sqrt(d)`, `x` to power `y`, etc: `math.pow(x,y)`
- These functions return float values in general – but integers can be used as floats (1.0, -3.0, etc).

Example: calculate total amount earned if invest *deposit* (eg, £100) for *n* years at interest rate *rate* (eg. 5%, or 0.05).

Calculating compound interest

deposit grows with interest to: $total = deposit * (1 + rate)^n$ after *n* years at rate *rate*.

```
import math
# The math library is needed for pow

amount = 100.0 # total money
deposit = 100.0 # original investment
rate = 0.05 # interest rate

for year in range(1,11) :
    amount = deposit * math.pow(1.0 + rate, year)
    print("After " + str(year) +
          " years, total is: " + str(amount))
```

Notice that comments are written following the # character.

Amounts after n years:

n	<i>amount</i>
1	105.0
2	110.25
3	115.763
4	121.551
5	127.628
6	134.01
7	140.71
8	147.746
9	155.133
10	162.889

Calculating compound interest

How many years does it take for investment to double?

Need `while` loop:

```
import math

amount = 100.0 # total money
deposit = 100.0 # original investment
rate = 0.05 # interest rate

year = 1
while amount < 2*deposit :
    amount = deposit * math.pow(1.0 + rate, year)
    print("After " + str(year) + " years, total is: " +
          str(amount))
    year = year + 1
```

while E:

 C

repeats *C* until *E* is false – but may run forever!

Here, termination when `year = 15`.

Try changing the rate and see the effect on the result.

What is Programming?

- Given: a *problem* (“find how many years it takes to double an investment at 5% interest”)
- First: *idea for solution*
- Second: *plan an algorithm* to carry out the solution
- Third: *code up algorithm* in a programming language (Python or another)
- Fourth: *test/correct* your code.

What is Programming?

- Problem to “find number of years until investment doubles”
- Idea for solution: calculate total amount earned for successive years, stopping when this amount is at least twice the original deposit
- Algorithm: compute total earned to *year* by

$$amount = deposit * (1.0 + rate)^{year}$$

for year = 1, 2, 3, ... stopping when $amount \geq 2 * deposit$

- Code in Python: use *while* loop to repeat calculation until the stopping condition is true
- Test with different rates and deposits.

Reading data from files

- Programs can read data from files (usually, text files)
- Eg.: to read student marks and classify these into grades
- Idea is that program reads lines of text from file (should be numbers), and converts each to a grade, until end of file (eof) is reached.

MarksFile.py:

```
file = open("marks.txt", "r")
for line in file :
    mark = int(line)
    if mark < 40 :
        result = "Fail"
    else :
        if mark >= 40 and mark < 50 :
            result = "Pass -- grade D"
        else :
            if mark >= 50 and mark < 60 :
                result = "Pass -- grade C"
            else :
                if mark >= 60 and mark < 70 :
                    result = "Pass -- grade B"
                else :
                    result = "Pass -- grade A"
```

```
print("The student's result is: " + result)
file.close()
```

The file *marks.txt* is opened for reading, and for each line of the file, its grade is calculated and displayed. Finally *marks.txt* is closed.

Defining functions

- So far we've written all code in one file as sequence of statements.
- Also possible to write several operations in a program, called from main code: factorial example.
- For larger programs, best to create separate files + call their code from main file.

Calculating factorials

For integer $n > 0$, its factorial is product: $n * (n - 1) * \dots * 1$.

Factorial.py:

```
def fact(n) :  
    if n <= 1 :  
        return 1  
    else :  
        return n*fact(n-1)  
  
# Main program uses fact function:  
  
for n in range(1,14) :  
    print("Factorial " + str(n) + "! = " + str(fact(n)))
```

fact calls itself – this is *recursion*. Provides another way of looping:
fact(5) calls *fact*(4), which calls *fact*(3), etc. **return e** ends the call and returns value of *e* to the caller.

Program design: lottery example

- Problem to “Carry out lottery with three numbers in range 1 to 30 for user to guess with five guesses”
- Idea for solution: generate random numbers, ask user to guess these, and count correct guesses
- Algorithm:
Generate 3 random integers in 1..30
Ask user for 5 guesses
Count and display the correct guesses
- Code in Python: use *random* package and `sample(range(1, 31), 3)` to generate the random numbers
- Test with different cases of correct/incorrect guesses.

Lottery.py:

```
import random # to generate random numbers

balls = random.sample(range(1,31),3)
correct = 0
for i in range(1,6) :
    sguess = input("Enter your guess: ")
    guess = int(sguess)
    if guess in balls :
        print("Correct guess of a ball")
        correct = correct + 1
    else :
        print("Sorry, wrong guess!")
print("You guessed " + str(correct) + " correct");
```

List variables

balls is a *list* variable, holds collection of values. Membership test

```
guess in balls
```

checks if value of *guess* is a member of *balls*.

- Previously, we saw variables that store single items – an individual integer, rational, string, or class instance
- Sometimes useful to have one variable storing group of items – such as the 3 balls in lottery game

- *List variables* store group of items of same kind. Eg.:

```
balls = [23,11,17]
```

declares *balls* as a list of 3 integers.

- Individual items are written as *balls*[0], *balls*[1], *balls*[2].

arrayX



```
arrayX = []  
arrayX.append(10)  
arrayX.append(20)  
arrayX.append(30)  
arrayX.append(40)  
arrayX.append(50)
```

or:

```
arrayX = [10,20,30,40,50]
```

Data in list variables

List variables

A loop of form

```
for x in array :  
    ... code for x ...
```

is often used to process lists.

Size of list *array* is $len(array)$. Notice that numbering of elements starts from 0, so ends at $len(array) - 1$.

Set value of $array[i]$ by statement

```
array[i] = value
```

Summary

- Program structure as list of statements
- Variables of integer, rational, string, class instance types
- List variables
- Assignment, conditional, loop statements
- Functions and recursion.

Further resources

We hope you have enjoyed this course. The following are useful for further study:

- Python can be downloaded from:
<http://python.org/downloads> Choose the option appropriate for your computer and operating systems.
- The examples used in this course are at:
www.nms.kcl.ac.uk/kevin.lano/c11