An Introduction to Programming with Python

Dr.-Ing. Norbert Völker
Many “computing devices” are embedded.
Can you think of computers/computing devices you may have in your home? Or which are used in shops?
Programs and Programming

- Programs determine what computers do.
  - “Software”
- Written by programmers, then “compiled” and “run” on the computer
- Can you think of some programs running on a computer or a mobile phone?

- Airbus A380 avionics: more than 100 million lines of code
  - Written by a large team of developers and engineers
Programming Languages

- There are (unfortunately) lots of them
  - Basic, C, C++, C#, Cobol, Java, JavaScript, Python, SQL, ...

- But luckily most of them share certain basic constructs/concepts
  - These are like the ABC of programming

- In our mini-course, we will use **Python 3**
  - It is a fairly beginner-friendly language

- Disclaimer: I am not a Python expert!
Main Topics

1) Simple values and operators: numbers, booleans and strings
2) Variables and assignments
3) Conditional Statements, random numbers and input
4) Loops
5) Lists and functions
1. Simple Values and Operators: Numbers, Booleans and Strings
Numbers and Arithmetic Operators

- Integer numbers: `int`
  - `42, 0, -301, ...`
- Floating point numbers: `float`
  - `7.92, 0.0, -12.0, 9e6`
- Four arithmetic operations: `+`, `-`, `*`, `/`
- Other numerical operators include `**`
  - Exponentiation (power)
  - `5 ** 3 == ?`

We can use the Python interpreter as a fancy calculator

```
(2+4)*(12-5)/3
```
Booleans and Comparison Operators

- Two logical truth values: **True, False**
  - Called “booleans”
- They come up frequently as the result of comparison operators (<, <=, ==, !=, >=, >):

  3 < 5
  4 == 3.1
  -1 != 77
Logical Operators

- not True is False
- not False is True

- (A and B)
  - is True, if both A and B are True
  - is False otherwise

- (A or B)
  - is True, if at least one of A and B is True
  - It is False if both A and B are False

What are the values of the following boolean expressions?

- (1 < 1) or (2 > 1) or (2 == 0)
- (2 != 1) and not(1 < 1 or 2 > 1)
Strings

- A **string** is a sequence of characters ("text")
- String literals are enclosed either in matching single or double quote symbols:
  - 'Hi!'
  - "Always look at the bright side"
  - "12"
- They can be concatenated using the + -operator
  - "Hello " + "Linda!"

- **Warning**: "5" is a string, not a number!
- There are functions to turn a string into a number:
  - `int("42")` → 42
  - `float("42")` → 42.0
2. Variables and Assignments

And `print()` Function
Variables and Assignments

- Non-trivial programs consist of a number of steps
  - How can we remember interim results?
- A variable has a name and it can be assigned a value
  
  ```
  x = 42
  y = x - 21.5
  moreThanHalf = y >= x/2
  name = "Sam"
  greeting = "Hello " + name + "!"
  ```
- A variable can be reassigned a new value: `x = x + 1`
- Exercise: determine the value of `a` and `b` at the end of the following program:
  
  ```
  a = 2;  b = 3;  a = b - 1;  b = 2*a
  ```
Breaking Down Arithmetic Calculations

- Calculations can often be simplified by breaking them down into a sequence of assignments.

- Example (derived from a 1968 O-Level Maths exam question)
  - A small factory makes 2700 golf balls each week
  - The raw material for each ball costs £1.35
  - Other expenses amount to £2850 per week
  - The balls are sold to a retailer at £2.50 each
  - Calculate the weekly profit

- How would you calculate this in an exam?
production = 2700
costPerItem = 1.35
fixedCost = 2850
salesPrice = 2.50
profit = production * (salesPrice - costPerItem) - fixedCost
print("Profit per week:", profit)

What are the advantages of solving this exercise as a Python program as compared to solving it on a calculator?
print() Function

- When we run Python programs from a file, it does not show output in the console by default.
- We can invoke the `print()` function in order to show informative messages.
- Examples:
  ```python
  print("Hello Linda!")
  print("Profit per week: ", profit)
  ```

- Each `print()` invocation starts a new line, and its output is normally all on one line.
- We can insert the special "newline character" \n in order to get line breaks:
  ```python
  print("First line
  Second line")
  ```
3. Conditional Statements, Random Numbers and Input
Control Flow

- **straight-line**
  - stmt1
    - stmt2
    - stmt3

- **conditional branching**
  - boolean
    - true
    - stmt1
    - stmt2
    - false
      - stmt2
      - stmt3
      - stmt4
Conditional Statement

- Evaluate a boolean expression:
  - If it is `true`, execute some statement(s)
  - If it is `false`, execute some other statement(s)
- Python syntax:
  ```python
  if ... :
    ...
  else:
    ...
  ```
- Special case: missing `else`-branch
  - Results in statement(s) *guarded* by a condition
- Python also has an `elif`-statement if you need to make a choice between more than two branches
Examples

if profit > 0:
    print("Factory is making a profit!")
else:
    print("Factory is not making a profit!")

if production < 0:
    print("Warning: production negative")
    print("Please check your numbers!")
(Pseudo-) Random Numbers

- Python has a function `randint(a, b)` that generates a random `int` value `x` such that `a <= x <= b`
  - `randint(1, 6)` returns a random integer between 1 and 6
  - Each invocation may produce a different value!
- Warning: these are not truly random numbers...
- The function is defined in a module called `random`
  - It has to be "imported" before it can be used
Rolling Two Dices

```python
from random import randint
dice1 = randint(1,6)
print("Player 1 dice:", dice1)
dice2 = randint(1,6)
print("Player 2 dice:", dice2)
if dice1 > dice2:
    print("Player 1 wins!")
elif dice2 > dice1:
    print("Player 2 wins!")
else: print ("Tie!")
```
Interactive Programs and `input()`

- Many programs are interactive. When they run, they
  - Request some input from the user
  - Respond with some output
- Function `input()` displays some text ("prompt") and then reads one line from the keyboard.
  - Program will block until user hits `return`- or `enter`-key
  - Result is returned as a string
- Typical usage
  ```python
  myVar = input("Please enter ...
  
  Do you remember the meaning of \n in a string?
  ```
- Use function `int()` or `float()` to convert to a number:
  ```python
  myNumber = float(input("Please enter ...
  ```
Calculating the (Length of the) Hypotenuse

- Give a right-angled triangle with hypotenuse $c$ and other sides $a$ and $b$

- Pythagoras Theorem:
  $$a^2 + b^2 = c^2$$

- Programming task:
  - Ask user for the length of sides $a$ and $b$
  - Calculate length of the hypotenuse $c$
  - Print result
import math
print("Hypotenuse calculation")
a = float(input(
    "Please enter length of first side:\n"))
b = float(input(
    "Please enter length of second side:\n"))
c = math.sqrt(a*a + b*b)
print("The length of the hypotenuse is", c)
4. Loops

*for*-loops
Number ranges
*while*-loops
**for-loop**

- Repeat a statement for all elements of a list:

```python
for i in [10, 20, 30]:
    print(i, "*", i, "=", i*i)
```

# Output:
10 * 10 = 100
20 * 20 = 400
30 * 30 = 900
**for-loop over a Range of Integers**

- `range(a, b)` returns a sequence of integers from `a` (inclusive) to `b` (exclusive)
  - `range(0, 6)` contains `0, 1, 2, 3, 4, 5`
- Integer ranges are often used in **for**-loops

```python
sum = 0
for i in range(1, 10):
    sum = sum + i
print("sum=", sum)
```
while Loops

- Iterate some statement(s) as long as a condition is fulfilled.

```plaintext
while ... :
 ...
```

- As the syntax suggests, the condition is checked before the statements in the body of the loop are carried out.
  - If the condition is `false` to start with, then the loop body is not executed.
while-Loop Example:
Throwing two dice until there is no tie

dice1 = 0
dice2 = 0
while dice1 == dice2:
    dice1 = randint(1, 6)
    dice2 = randint(1, 6)
print("Player 1 dice:", dice1)
print("Player 2 dice:", dice2)
if dice1 > dice2:
    print("Player 1 wins!")
elif dice2 > dice1:
    print("Player 2 wins!")
else: print("Tie!")
Interactive Loops

- Remember our hypotenuse-computation program
  - What if the user wants to repeat this computation?
- They could restart the program but that would be a bit cumbersome
- Solution: interactive loop
  - Repeat computation as long as user wants it
- Problem: how to find out if user wants to stop?
- Simple approach:
  ```python
  print("Do you want to continue (y/n)?")
  stop = input().lower().startswith("n")
  ```
- Can you guess what functions `lower` and `startswith` do?
import math

print("Hypotenuse calculation")
stop = False
while (not stop):
    a = float(input("Length of first side?\n"))
    b = float(input("Length of second side?\n"))
    c = math.sqrt(a*a + b*b)
    print("Length of hypotenuse =", c)
    print("Do you want to continue (y/n)?")
    stop = input().lower().startswith("n")
print("Good-by!")
5. Lists and Functions

Also reading data from a file
Lists

- One of Python's inbuilt sequence data structures
- Values ("items") are separated by commas and enclosed in square brackets
  ```python
  myList = [2,3,5,8,13,21,33]
  ```

- Elements can be accessed by indexing
  - Warning: first element has index 0
- Example:
  ```python
  myList[6] = 34
  ```
Some Sequence Operations

- \( s[i] \) \( i \)th item of \( s \), origin 0
- \( s[i:j] \) slice of \( s \) from \( i \) (inclusive) to \( j \) (exclusive)
- \( \text{len}(s) \) length of \( s \)

- These operations work on lists and on strings. Example

```python
s = "abcdefg"
print(s[2])
print(s[3:5])
print(len(s[1:5]))
```

- Python comes with many useful functions and operators on lists, please see literature
Reading Data from a File

- Open file
  
  ```python
  f = open ("m:/myFile.txt");
  ```

- Reading lines one by one:

  ```python
  for line in f:
      ...
      # code that does something with line
  ```

- You can even read files directly from the internet
  - Call function `urlopen()` instead of `open()`
  - See example below
Example: Summing up Numbers from an Online File

```python
from urllib.request import urlopen
url = ('http://orb.essex.ac.uk/ce/ce832' +
       '/schoolVisit2015/numbers.txt')
print("Reading numbers from:", url)
data = urlopen(url)
sum = 0
for line in data:
    sum += float(line)
print("Sum of numbers is", sum)
```
Using Existing Functions

- Python comes with lots of functions
  - Some are built-in and immediately available
    - `len()`, `min()`, `max()`, `sorted()`, `sum()`, ...
  - Other functions are organised in modules
    - these need to be `imported` before you can use them
- We have already seen two examples of this
  - module `random`: function `randint()`
  - module `urllib.request`: function `urlopen`
- It is good practice to re-use existing functions
  - no need to code everything yourself
Function Definition Example: hypotenuse()

Define a function which computes the length of the hypotenuse in a right-angled triangle given the length of the two other sides.

```python
import math
def hypotenuse(a,b):
    return math.sqrt(a*a + b*b)

# Usage example

for i in [3,4]:
    for j in [4,5]:
        print("hypotenuse(",i,",",j,"\) = ",
              hypotenuse(i,j))
```
Concluding Remarks

- We have covered a lot of ground in this lecture
  - In our Year 1 programming module CE151, these topics are taught during several weeks of teaching!
- For more details on Python, please see online documentation and tutorials
- We will do some exercises in the lab sessions
  - There will be several staff members to help you
  - Please ask questions if you get stuck!
- Programming is a craft
  - It requires practice