UCL

CEGE0096: Geospatial Programming

Lecture 1: Introduction

September 30, 2019

Aldo Lipani

Welcome!

CEGE0096

Geospatial Programming

Tutor: Dr. Aldo Lipani (aldo.lipani@ucl.ac.uk)

Use Moodle forum for questions!

https://moodle.ucl.ac.uk

This course provides an **introduction** to **programming** in the context of **geospatial science** and **technology**.

Aims and Learning Outcomes

By the end of the module, you will:

- Understand the basic principles of **imperative programming**
- Understand the basic principles of **object-oriented programming**
- Automate geospatial analysis workflows and map production
- Become familiar with:

Anaconda

Python

Python Notebooks

PyCharm

Git

Fast Paced Course

Position yourself to succeed!

- **Read the assignments when they come out** and come back to them later
- The practicals (after the assignments are released) are dedicated to improve skills useful to solve them

New to programming? **PRACTICE. PRACTICE? PRACTICE!**

- You can't passively absorb programming as a skill
- Run the python notebooks yourself, play with the code
- Do the exercises
- Experiment with new Python commands!

Experienced programmer? **HELP. HELP? HELP!**

- By helping your colleagues you will consolidate your knowledge and get better in solving the assignments

Course Structure – Part I

Week	Date	AM/PM	Content	Room Type
1	30 Sep	PM	Introduction (Programming and Python)	Lecture
	3 Oct	AM	Anaconda, Python Notebooks and Python Basics	Practical
2	7 Oct	PM	Python (Continue), IDE and Debugger	Lecture
	10 Oct	AM	PyCharm, Debugger and Python	Practical
3	14 Oct	PM	Python Data Structures and Version Control	Lecture
	17 Oct	AM	Git, GitHub and Problem Solving	Practical
4	21 Oct	PM	Object Oriented Programming and GIS Data Structures with Python	Lecture
	24 Oct	AM	Drawing with Python	Practical
5	28 Oct	AM	The Individual Project is Released	
	28 Oct	PM	Geometry with Python	Lecture
	31 Oct	AM	Review	Practical
6	Reading Week			
	10 Nov	PM	Individual Project Deadline (11:59 PM)	

Course Structure – Part II

Week	Date	AM/PM	Content	Room Type
1	11 Nov	PM	Trees and Graphs	Lecture
	14 Nov	AM	To Define	Practical
2	18 Nov	PM	To Define	Lecture
	21 Nov	AM	To Define	Practical
3	25 Nov	PM	Working with Real Data in Python	Lecture
	28 Nov	AM	To Define	Practical
4	2 Dec	AM	The Group Project is Released	
	2 Dec	PM	To Define	Lecture
	5 Dec	AM	To Define	Practical
5	9 Dec	PM	Copyright and GDPR	Lecture
	12 Dec	AM	Review	Practical
	13 Jan	PM	Group Project Deadline (11:59 PM)	

Course Venues

- Lectures are at 2 PM on Mondays,
- Practicals are at 11 AM on Thursdays,
- Use campus route finder if you get lost (https://www.ucl.ac.uk/maps):

This Thursday:
Christopher Ingold Building G20

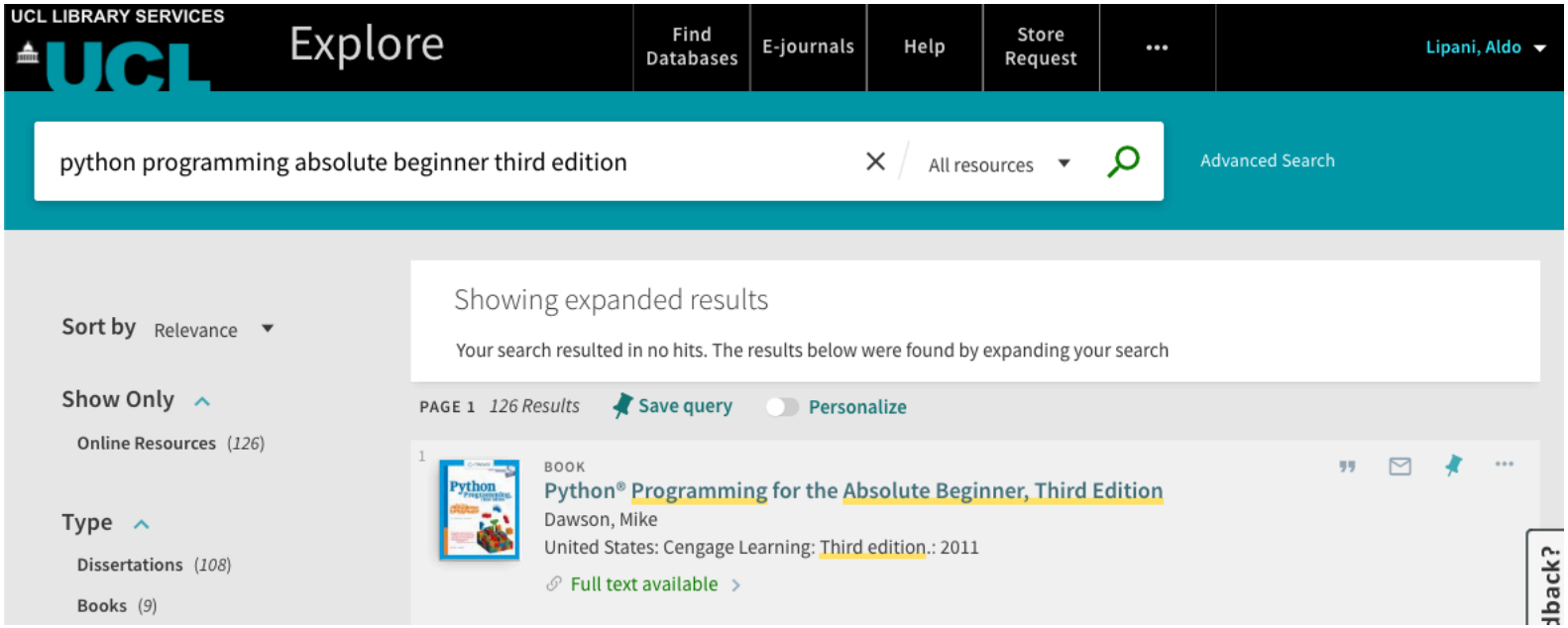
WARNING: Practicals rooms change often! Always check where it is!

Assessments

- 1 individual programming assignment (50%)
  - Geometric problem
- 1 group programming assignment (50%)
  - GIS problem
  - Using real data
- In both assignments you will be evaluated based on:
  - The solution you found
  - The use of git
  - A report
  - Etc. (details will be provided in the assignments sheets)

Recommended Readings

- From Moodle (https://ucl.rl.talis.com/modules/cege0096.html)



- For the first assignment this book and the course material will suffice

PROGRAMMING

What is Programming?

- It's a way to tell a computer what to do.

- It is done via a programming language:
  - It's a set of rules for communicating an **algorithm**
  - It provides a linguistic framework for describing computations

- Enables complex data handling and processing

- Not solely the pursuit of specialists: programming lies at the heart of almost everything we do.

What is Algorithm?

- Algorithms resemble recipes. They take “inputs” (the ingredients), perform a set of simple steps, and then terminate after producing an “output” (the meal)

- But algorithms are not like recipes because their steps are well-defined and expressed in a precise way (1/2 cup!?)

- An algorithm is a precise description of sequence of steps that given an input terminates returning an output

Why Programming?

- Teaches you how to break down a complex problem and construct methods for achieving a solution – **computational thinking**
- Allows you to access a wide range of already **developed tools**
- Makes you **more employable**

You will work with multiple large geospatial (or spatio-temporal) datasets:

- Processing
- Analysis
- Visualization and Mapping

… but what about GUI-based software, e.g. Excel, ArcMap?

Programming vs. GUI-based Software

GUI-based software can only take you so far:

- You can create a map of restaurants in a town,
  but what if you need the same map for every town in the UK?
- You can assign a missing coordinate reference system to a single dataset when you load it in GIS application,
  but what if you have thousands of datasets with missing CRS?
- You can carry out Kriging using a toolbox of GIS applications,
  but what if the parameters in the tool don't give you the flexibility you need? What if you need to analyse/visualise the results in a different way?

GUI-based software will only allow you to do what the developer has **programmed**.

Programming allows you to tell a computer what you want.

Programming Languages

A **programming language** is a formal language that is used to instruct a computer how to perform tasks

There are many different programming languages. Each one is designed to suit different applications and users:

- High performance?
- Portable?
- Flexible?
- Rapid development?
- Graphical?

Which programming language?

Levels of Programming Languages

High-level program

```
class Line2D:

    def __init__(self, x1, y1, x2, y2):
        self._x1 = x1
        self._y1 = y1
        self._x2 = x2
        self._y1 = y2
```

Low-level program

```
mov rax, 1
mov rdi, 1
mov rsi, message
mov rdx, 13
syscall
mov rax, 60
xor rdi, rdi
```

Executable Machine code

```
00101000100010001001111001010001000100010
01000101000100100011111100101010010010010
01000000100111101001001001000101010010100
10010010100011111111111110000000101010100
00100000000010101010001001001001000100100
10010010100101010010101000100010100100100
10011111000100100101001000000000000000000
```

Example of Language Types

More sophisticated, restricted-purpose languages

- MATLAB, R, LaTeX

High-level multi-purpose languages

- Python, Ruby, Perl

Mid-level multi-purpose languages, enabling greater flexibility and performance

- Java, C/C++, C#

Low-level machine code

- Assembly language

Web development languages

- JavaScript, XML, HTML, PHP

Graphical languages

- Scratch, FME

Language Paradigms

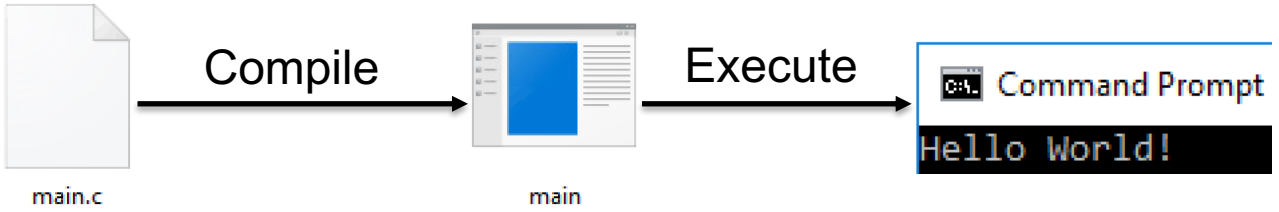
A programming paradigm is a style, or “way” of programming.

Paradigms:

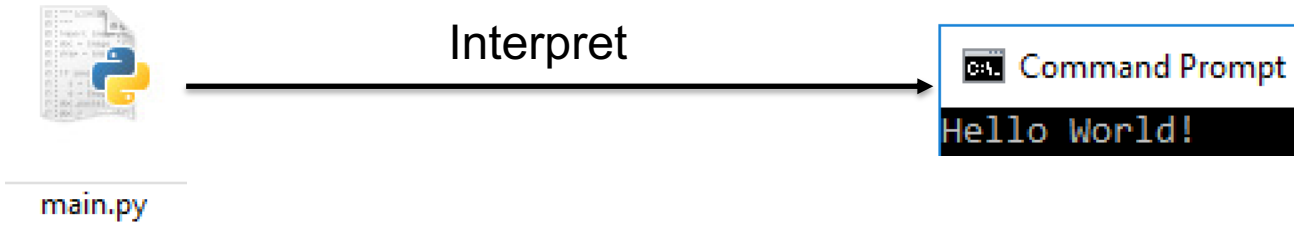
- **Imperative**
  - Results are achieved by ordered change of state
- Functional
  - … by a series of function applications
- Logic
  - … by declaring a question about a system of facts and rules
- **Object-oriented**
  - … by grouping instructions and states together
- Hybrid

Compiled vs. Interpreted Languages

Programs written into compiled languages are first **compiled** (translated) into machine code:



Programs written into interpreted languages are **interpreted** into machine instructions:



What Language is the Best?

IEEE SPECTRUM

Rank	Language	Type	Score
1	Python		100.0
2	Java		96.3
3	C		94.4
4	C++		87.5
5	R		81.5
6	JavaScript		79.4
7	C#		74.5
8	Matlab		70.6
9	Swift		69.1
10	Go		68.0

Python. (Next Slide)

Java. OOP, “compiled”, runs on JVM

C. compiled, fast, (too) flexible

C++. OOP, like C

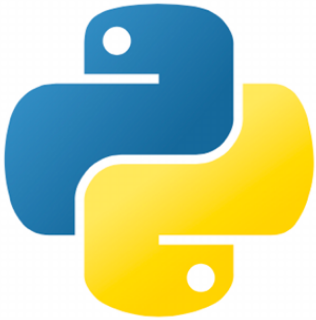
R. OOP, functional, interpreted, slow

JS. OOP, interpreted, runs on browsers

…

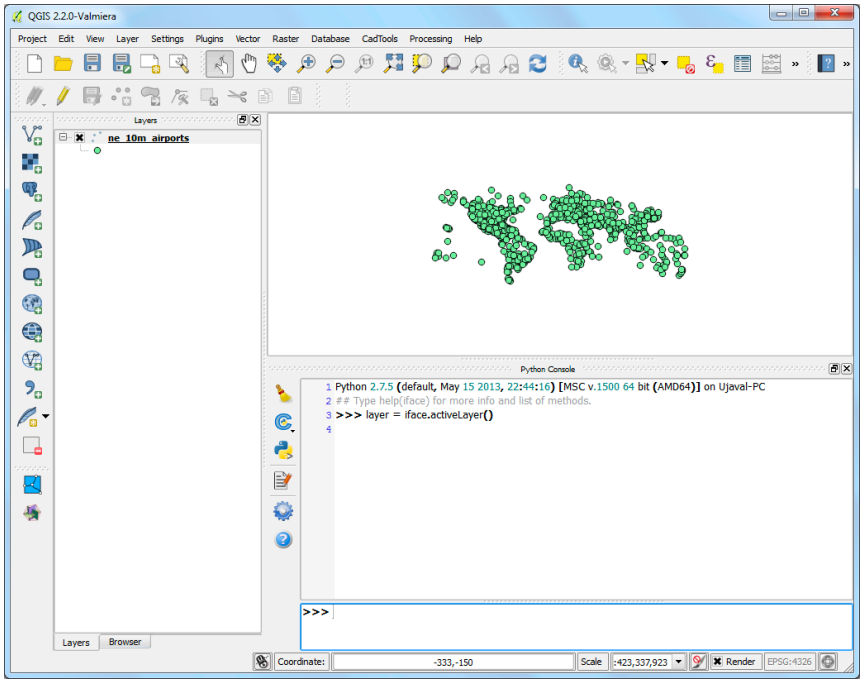
Python

- Python was created by Guido van Rossum in 1989.
- Python is quick:
  - To learn
  - To write
  - To read
  - To debug
  - To maintain
- Python, today, has a large community of enthusiasts, who create useful packages for data science, statistics, machine learning, etc.
- Python is multi-paradigm:
  imperative, object-oriented and functional.
- Python is an interpreted language.



Python for Geospatial

- Lots of libraries to work with geographic data
- Used for scripting and plugin building in QGIS and ArcGIS



Examples of Mapping in Python

Python can be used for mapping directly

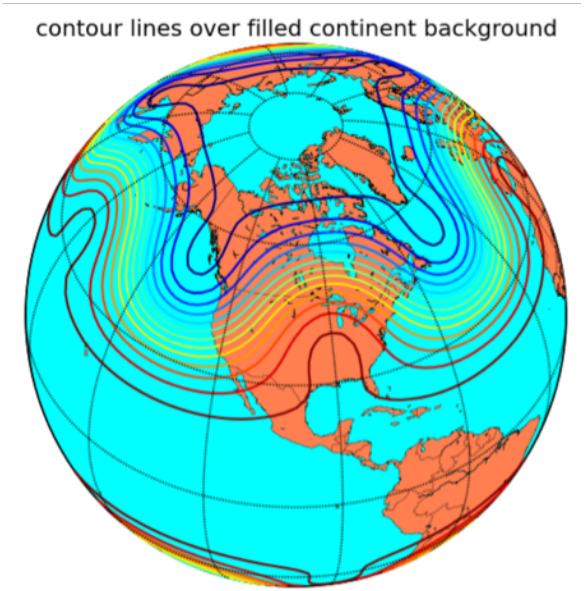
matplotlib is a powerful library for data visualization

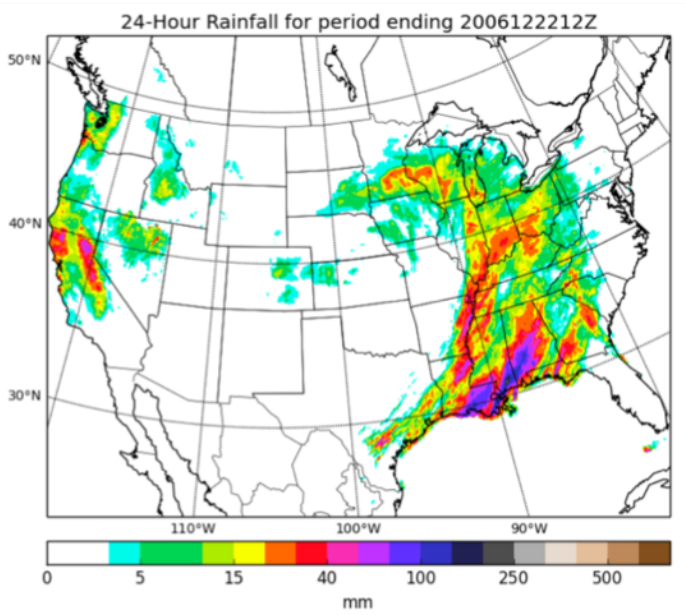
Many libraries for dealing with geospatial data:

- GeoPandas
- shapely
- rasterio
- pyproj
- PySAL
- …

Examples of Mapping in Python (2)

These two visualizations were made using Python alone (no GIS software)





PYTHON

Hello, World!

1. Open the python interpreter or
   … insert a code cell in a Jupiter notebook
   … create a new python script in an IDE

2. Write

```
print("Hello, World!")
```

3. Hit return or
   … press the play button/shift+return
   … run the script

4. Observe your computer greeting the World!

Our Python (Imperative) Journey

- Types
- Operators
- Variables
- Simple I/O
- Branching
- Loops
- Program Planning
- Strings
- Tuples
- Lists
- Dictionaries
- Files

Types (I)

Integers (whole number):

- Named `int`
- Range $(-\infty; \infty)$
- You can express integers in various bases:

```
print(58, 0b111010, 0o72, 0x3A)
```

Floating-Point Numbers (real numbers)

- Named `float`
- Range $(-\infty; \infty)$ (virtually)
  - if greater than `1.79e308` it becomes `inf`
  - if smaller than `5e-324` it becomes `0`

```
print(3.1415)
```

Types (II)

Complex Numbers:

- Named `complex`
- Like `float`
- Use `.real` or `.imag` to extract the real or imaginary parts

```
print(1+5j, 1+5j.real, 1+5j.imag)
```

Boolean:

- Named `bool`
- Values `True` and `False`

```
print(True, False)
```

Types (III)

Strings (sequences of character data):

- Named `str`
- As long as you wish
- Delimited by `""` or `''` (but please be consistent)
- Escape with `\` to insert `'` or `"` in your string

```
print("I am a string.")
```

- Use `"""` to write escaped strings:

```
print("""This is a
string that spans
across several lines""")
```

Operators (I)

With numeric types, **arithmetic operators**:

Operator	Description	Example	Result
+	Addition	`13+2`	`15`
-	Subtraction	`13-2`	`9`
*	Multiplication	`13*2`	`36`
/	Division	`13/2`	`6.5`
%	Modulus	`13%2`	`1`
//	Floor Division	`13//2`	`6`
**	Exponent	`13**2`	`169`

WARNING: Strange results when applied to Boolean types

Operators (II)

With numeric or Boolean types, **comparison operators**:

Operator	Description	Example	Result
==	Equality	`13==2`	`False`
!=	Inequality	`13!=2`	`True`
>	Greater	`13>2`	`True`
<	Less	`13<2`	`False`
>=	Greater or Equal	`13>=2`	`True`
<=	Less or Equal	`13<=2`	`False`

Operators (III)

With Boolean type, **logical operators**:

Operator	Example	Result
and	`True and False`	`False`
or	`True or False`	`True`
not	`not True`	`False`

WARNING: Strange results when applied to numeric types

Operators (IV)

With String type, **string operators**:

Operator	Description	Example	Result
+	Concatenation	`"hello" + "world"`	`"helloworld"`
*	Repetition	`"hello" * 5`	`"hellohellohellohel`
in	Membership	`"hello" in "world"`	`False`
not in	Not Membership	`"hello" not in "world"`	`True`
[]	Slice	`"hello"[1]`	`"e"`
[:]	Range Slice	`"hello"[0:2]`	`"he"`

What happens if to the slice operator you pass a negative number?

So far

You can use Python as a powerful calculator

By combining data types and operators,
you can write:

```
12 * 10 + 32 % 2 / 2 - 78
```


```
print("hello"[-4] + "and "[1:4] + "ehthe "[2:6] + "world")
```

Variables

- A variable provides a way to label and access information
- You can create a variable by typing:

```
name_of_the_variable = "and its value"
```

- Once created, you can access its content by using it as data

```
name_of_the_variable * 5
```

```
name_of_the_variable[2:4]
```

```
print(name_of_the_variable)
```

Naming Variables

- Names of variables are case sensitive and cannot **start** with a number. They can contain letters, numbers, and underscores.

```
bob   Bob   _bob   _2_bob_   bob_2   BoB
```

- There are some reserved words:

```
and, assert, break, class, continue, def, del, elif, else,
except, exec, finally, for, from, global, if, import, in,
is, lambda, not, or, pass, print, raise, return, try, while
```

Simple I/O

- You have already seen `print` in action
- It prints to screen the content of a variable or the result of a calculation
- "`print`" is a built-in function
- You can pass multiple strings, these will be concatenated with a space

- To read from keyboard you can use the `input` function
- It waits for the user to type something on the console until the user presses `enter`
- Then it returns the `string` value inputted by the user
- To convert it to a digit use the function `int` (or `float`).

```
user_input = input("Please type something: ")
print("User input:", user_input)
```

Converting Types

- The built-in `str` function converts a data type into a string
- You can convert a string to an integer using `int`
- You can convert a string to a floating-point number with `float`

- Example of conversion of an integer to a string:

```
x = 3
print("x is equal to " + str(x))
```

- Example of conversion of a string to a float:

```
pi = input("insert pi ")
print(pi) # "3.14"
pi = float(pi)
```

So far

You can use Python as a more powerful calculator

```
h0 = "rabbit"
h1 = h0
h0 = 5
print(h0, h1)
```

Where is the rabbit?

Flow Control

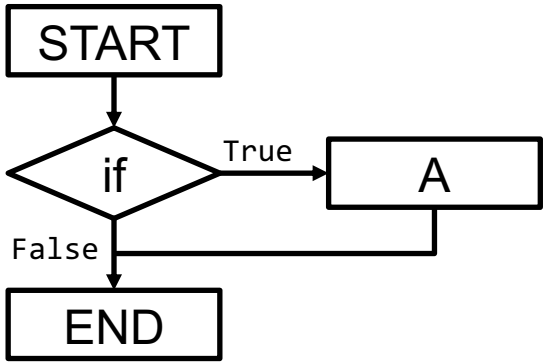
- Control statements are the building blocks of computer programs
- Allow application of conditions on actions
- Allow repeated execution of actions on multiple data

- Three key types of flow control
  - if condition statements
  - while loops
  - for loops

If Statement (I)

If the condition is verified the block below the if statement is executed

```
if x == 42:
    print("x is equal to 42")
```



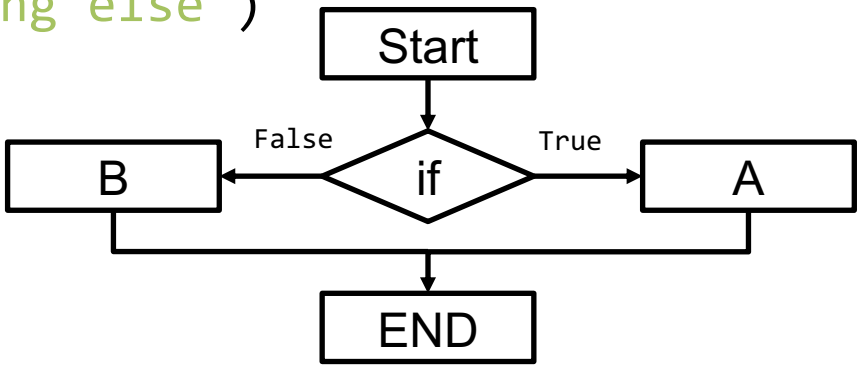
Note:

- Use of indentation for blocks
- Colon (:) after Boolean expressions

If Statement (II)

If the condition is not verified the else block is executed

```
if x == 42:
    print("x is equal to 42")
else:
    print("x is equal to something else")
```



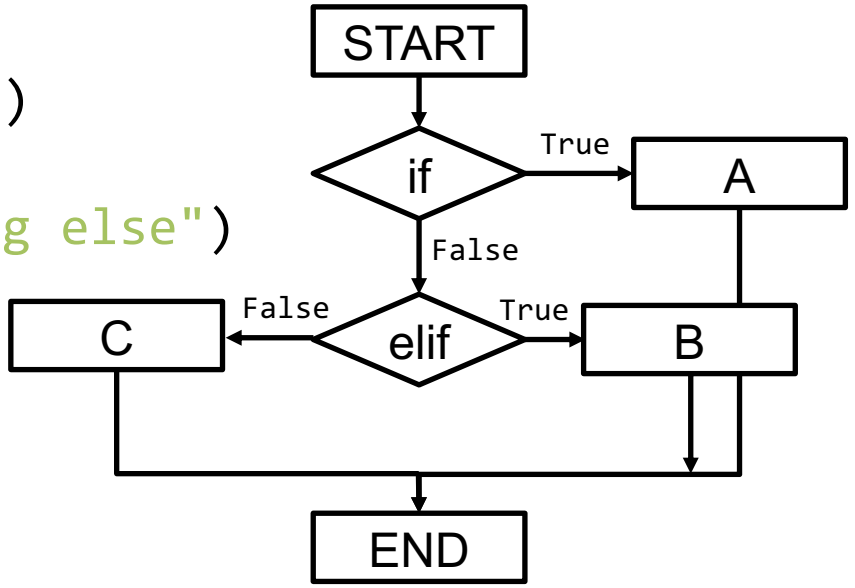
Note:

- Use of indentation for blocks
- Colon (:) after Boolean expressions

If Statement (III)

If you have multiple conditions to check, you can use the elif block. If the if condition is not verified and the elif block is verified then execute the elif block.

```
if x == 42:
    print("x is equal to 42")
elif x == "hello":
    print("x is equal to 'hello'")
else:
    print("x is equal to something else")
```



Note:

- Use of indentation for blocks
- Colon (:) after Boolean expressions

IMPORTANT: Whitespace

- Whitespace is meaningful in Python: especially indentation and placement of new lines (i.e. press Enter on keyboard).
- Use a newline to end a line of code.
    - Use \ when must go to next line prematurely.
- No braces { } to mark blocks of code in Python…
  Use consistent indentation instead.
    - The first line with more indentation starts a nested block
    - The first line with less indentation is outside of the block.
- A colon appears at the start of a new block
- Convention is **4 spaces** or **1 tab**, but **don't mix them up**!

IMPORTANT: Whitespace

Whitespace Example

```
if type(x) == int:                    # Level 1
    print(x, "is an integer")         # Level 2
    if x == 0:                        # Level 2
        print(x, "is zero")           # Level 3
else:                                 # Level 1
    print(x, "is not an integer")     # Level 2
```

- No braces { } to mark blocks of code in Python…
  Use consistent indentation instead.
  - The first line with more indentation starts a nested block
  - The first line with less indentation is outside of the block.
- Convention is **4 spaces or 1 tab** but can be any number as long is it is consistent; **don't mix them up**!

Exercise (10 minutes)

Write a program that requests the user to `input` a temperature in degree Celsius and `print`s out a suitable message according to the temperature state below:

- “Freezing” when the temperature is below zero;
- “Very cold” when it’s between 0 and 10 (not included);
- “Cold” when it’s between 10 and 20 (not included);
- “Normal” when it’s between 20 and 30 (not included);
- “Hot” when it’s between 30 and 40 (not included);
- “Very hot” when it’s above 40.

You are allowed to use variables, operators and if statements.

Function reminder:

`input("message")` prints the message and waits the user for an input

`float` to convert strings to floating-point numbers

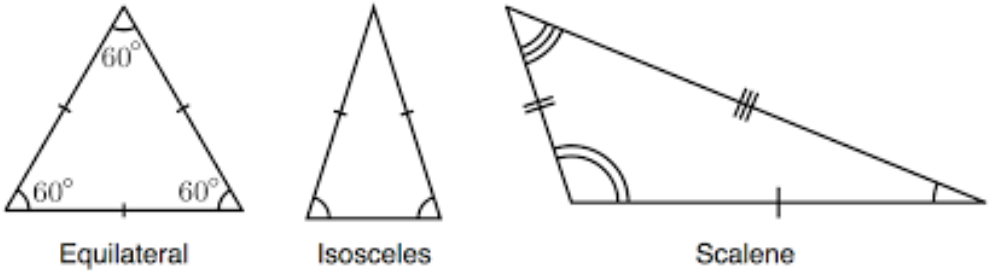
Solution

```
print("Temperature to Perceived Temperature Converter (0.1)")

temp = input("What temperature is outside? ") # This returns a string
temp = float(temp)                            # This converts it to a float

perceived_temp = ""
if temp < 0:
    perceived_temp = "freezing"
elif temp < 10:
    perceived_temp = "very cold"
elif temp < 20:
    perceived_temp = "cold"
elif temp < 30:
    perceived_temp = "normal"
elif temp < 40:
    perceived_temp = "hot"
else:
    perceived_temp = "very hot"

print("Your temperature today is", perceived_temp)
```

Exercise (10 minutes)

Write a program that ask the user for 3 angles in degrees and checks, if possible, which triangle you can build with them:

- Equilateral triangles have all angles equal;
- Isosceles triangles have 2 angles equal;
- Scalene triangles have all of them are different.



Rules are the same of the previous exercise.

Solution

```
print("Angles Checkers for Triangles (0.1)")

angle1 = float(input("1st angle: "))
angle2 = float(input("2nd angle: "))
angle3 = float(input("3rd angle: "))

if angle1 + angle2 + angle3 != 180:
    print("You can't build any triangle out of these angles!!!")
elif angle1 == angle2 and angle2 == angle3:
    print("You can build an equilateral triangle")
elif angle1 == angle2 or angle2 == angle3 or angle3 == angle1:
    print("You can build an isosceles triangle")
else:
    print("You can build a scalene triangle")
```

Summary

```
x = 34 - 23
y = "Hello"
z = 3.45
if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + "World"
print(x)
print(y)
```

```
# A comment.
# Another one.
```

```
# String concatenation
```

1. A comment
2. Variable assignment
3. Whitespace
4. A mathematical calculation
5. A control statement
6. Function call

See you on Thursday for the Practical

Christopher Ingold Building G20