6 | INTRODUCTION TO PYTHON

Introduction

This notebook starts with a brief overview of languages and coding environments, before introducing basic coding concepts in the Python languages, including code comments, simple arithmetic, and the control of program flow.

A computer program

A computer program is a sequence of written (typed) instructions that allows a computer to perform specified tasks.

A computer program is written in one (or more) computer languages. Such languages can be classified in many ways. One such classification compares **compiled languages** to **interpreted languages**. In most languages, the instructions, called the **source code**, is written in human reaable form. This source code is then either **compiled** (translated) into computer instruction specific to a computer type or it is simply **interpreted**.

In the case of interpreted languages, another program called an interpreter, interprets the language code for execution. Python and R are examples of interpreted languages and Julia is an example of a compiled language.

Python

Python is a general-purpose programming language developed by Dutch computer scientist Guido van Rossum. He began work on Python in the late 1980's and published the first version in 1991.

While Python can be used in many programming applications such as game development, app development, web development, and much more, it is arguable best known as the principal language for data analysis, data science, statistics, biostatistics, and machine learning.

Python constantly ranks amongst the most used programming languages. The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month.

Python is an interpreted language as defined above. Lines of code are individually translated to code that a computer can understand and then executed. Since speed is a major benefit of compiled languages, it bares mentioning that Python is slow to execute. Why then, is Python so popular?

There is no doubt that the popularity of Python is rooted in its ease of use. It is remarkable easy to learn Python. The simple nature of the syntax means that it reads almost like English sentences. It is very easy to translate an idea into computer code using Python. The translation from idea to code is referred to as **computational thinking**. Computational thinking is a core skill in research today and applies to the use of any computer language.

Task: Read the Wikipedia) page on the Python programming language. Follow some of the links in the first few paragraphs to learn more about technical computer language terms such as *high-level language*, *code readability*, the use of *significant indentation*, *dynamically-typed languages*, *garbage-collection*, *multiple programming paradigms*, *object-oriented programming*, and *functional programming*.

Coding environments

Computer code is written in a coding environment. Coding environments range from the bare-bones REPL (read-evaluate-print-loop) terminal or command line interfaces, to rich development environments such as PyCharm, Visual Studio Code, and many more.

These notes are created using a modern approach to computing, called a notebook. Many language include the ability to create notebooks. The Wolfram Language was the first langauge to make use of this and the language's creator, Stephen Wolfram, calls these **computational essays**. This document and all the documents that we will use in this class are Jupyter notebooks.

1. **Jupyter notebooks** started life as IPython notebooks. IPython is a particular *flavor* of Python and stands for **interactive Python**. In interactive Python, we write a few lines of code called scripts and execute then one at a time. It is ideal for exploration, especially for data exploration.

2. **Ipython notebooks** were created for the Python language. Today we use this notebook environment for other languages such as Julia and R, hence the name change to Jupyter notebooks.

3. Jupyter notebooks are being replaced by **Jupyter Lab**. Jupyter Lab is a more comprehensive environment that includes notebooks as a part of other useful coding tools. Jupyter notebooks or Jupyter Lab runs in a web browser. Many coding environments such as PyCharm, DataSpell, Visual Studio Code, and more can also generate Jupyter notebooks.

As the term *notebook* suggest, these computational essays allow us to write normal sentences and paragraphs, with styling such as headings and subheadings, the inclusion of images, sounds, videos (such as YouTube), files, and more.

Code can also be included in a notebook. The code is written as short scripts and are executed immediately, with the result displayed underneath the code.

Lines of code are written in Code cells. Text, images, and such are entered in Markdown cells. A Jupyter notebook consists of many cells. Either a Code cell or a Markdown cell can be created after or between any cells.

Below is a Code cell, with the result of the code displayed following the cell.

In [1]:
```
import numpy
import pandas
from plotly import express
from plotly import io

io.templates.default = 'ggplot2'

age = numpy.round(
    numpy.random.normal(
        loc=50,
        scale=10,
        size=100
    ),
    decimals=0
)

dbp = numpy.round(
    (0.8 * age) + numpy.random.normal(
        loc=50,
        scale=10,
        size=100
    ),
    decimals=0
)

df = pandas.DataFrame(
    {'Age':age,
     'Diastolic BP':dbp,
     'Treatment group':numpy.where(dbp>90, 'Placebo', 'Treatment')}
)

express.scatter(
    df,
    x='Age',
    y='Diastolic BP',
    color='Treatment group',
    hover_name='Treatment group',
    title='Diastolic blood pressure as a function of age'
).update_traces(
    marker=dict(
        size=12,
        opacity=0.8,
        line=dict(
            width=2,
            color='DarkSlateGrey'
        )
    ),
    selector=dict(
        mode='markers'
    )
)
```

Python 101

Code comments

In most cases, we write code to review it later or to share it with others. In both cases, it makes sense to leave comments about the code. This will remind you what your code is doing when you review it later and help others understand what you tried to achieve.

When we write lines of code, we can prepend any line (or part of a line) uing the pound or hashtag symbol, `#`. The Python interpreter will ignore everyting in a line following the `#` symbol. It is therefor ideal to use when leaving comments on our code.

In most of the code cells below, you wil notice the use of code comments.

Arithmetic operators

We are all familiar with the basic arithmetical operators such as addition and multiplication. The simple use of mathematical operations in Python are examples of **expressions**. The actual symbols such as `+` and `-` are termed **operators**. The table below shows a list of these operators.

Expression	Operator	Example	Result
Adding	`+`	`2 + 2`	`4`
Subtracting	`-`	`8 - 3`	`5`
Multiplying	`*`	`2 * 2`	`4`
Dividing	`/`	`8 / 4`	`2`
Integer devision	`//`	`10 // 3`	`3`
The remainder	`%`	`10 % 3`	`1`
Exponentiation	`**`	`2 ** 4`	`8`

Addition

Adding $2+2$, which should result in 4. Holding down the SHFT key and then hitting ENTER (Windows and Linux) or RETURN (Mac) excutes a cell. There is also a button above each cell in many coding environments that can be clicked.

```
In [2]: # Add 2 and 2
        2 + 2
```

```
Out[2]: 4
```

Note the use of spaces between the 2 and the $+$ symbol. This is simply for ease of reading. The spaces can be omitted, writing `2+2`. Note also the use of a code comment. A code comment is started by a pound symbol (or hashtag). Python ignores any script following a pound symbol (for a specific line of code).

More than two numbers can be added in a single expression. Below, 2 and 2 and 10 are added.

```
In [3]: # Add 2 and 2 and 10
        2 + 2 + 10
```

```
Out[3]: 14
```

Subtraction

One number is subtracted from another using the `-` operator.

```
In [5]: # Subtract 3 from 10
        10 - 3
```

```
Out[5]: 7
```

More than one number can be used in a subtraction expression.

```
In [6]: # Subtract 2 and 3 from 10
        10 - 2 - 3
```

```
Out[6]: 5
```

Multiplication

Since keyboards do not have a multiplication key, the `*` symbol is used for this operation. It is ususally a part of the `8` key.

```
In [7]: # Multiply 2 by 3
        2 * 3
```

```
Out[7]: 6
```

More than two numbers can be multiplied.

```
In [8]: # Multiply 2 and 10 and 3
        2 * 10 * 3
```

```
Out[8]: 60
```

Division

As with multiplication, a special key is used since there is no $\div$ key on a standard keyboard. The forward slash key, `/` is used for division.

```
In [9]: # Divided 10 by 2
        10 / 2
```

```
Out[9]: 5.0
```

Note the use of the integers 10 and 2 above. The result, though, is 5.0, decimal value, known in Python as a floating point value. The result of a division is always a floating point value.

The result of dividing 20 by 8 results in a value with a decimal point. The `//` operator can be used to return only the whole number (integer) part of the solution.

```
In [10]: # Use integer division to divide 20 by 8
         20 // 8
```

```
Out[10]: 2
```

The result is 2, the whole number part of the solution, with $2 \times 8 + 4 = 20$. The remainder operator, `%` can be used to return the remainder of a division.

```
In [11]: # Return the remainder of 20 divided by 8
         20 % 8
```

```
Out[11]: 4
```

Powers

Note that $2^3 = 2 \times 2 \times 2 = 8$. The `**` operator is used to raise a number to a power.

```
In [12]: # Raise 2 to the power of 3
         2 ** 3
```

```
Out[12]: 8
```

Order of arithmetical operations

There is an order to mathematical operations, i.e. division and multiplication comes before subtraction and addition. In the expression $3 + 4 \times 2$, the 4 and 2 and multiplied first resulting in 8. The 2 is then added to yield 11.

```
In [13]: # Add 3 and 4 and multiply by 2
         3 + 4 * 2

Out[13]: 11
```

Parentheses can be used to change the order of operations. In the expression $(3 + 4) \times 2$, the 3 and 4 are added first, resulting in 7. The 7 is then multiplied by 2 to yield 14.

```
In [14]: # First add 3 and 4, then multiply by 2
         (3 + 4) * 2

Out[14]: 14
```

Task: Calculate a solution to the expression below.

$$\frac{(6+4) \times 2}{10}$$

Comparison operators

Comparison operators or **conditionals** are used to return the value of a comparison. The return is one of two Boolean values: `True` or `False` based on the comparison being made.

The two **Boolean** values are `True` and `False`. The data type of these are printed below using the `type` function. See below under Python data types for more information on the `type` function and data types in Python.

```
In [15]: # Data type of True
         type(True)

Out[15]: bool
```

```
In [16]: # Data type of False
         type(False)

Out[16]: bool
```

The Python data type of `True` and `False` are both `bool`. Internally, `True` is stored as the integer 1 and `False` as the integer 0. Arithmetic can be done with the values.

```
In [17]: # Add True and True
         True + True

Out[17]: 2
```

```
In [18]: # Add True and False
         True + False

Out[18]: 1
```

The Python comparison operators are listed in the table below.

Comparison	Operator	True example	False Example
Less than	<	2 < 4	4 < 2
Greater than	>	4 > 2	2 > 4
Less than or equal to	<=	4 <= 4	2 <= 4
Greater than or equal to	>=	4 >= 4	2 >= 4
Is equal to	==	4 == 4.0	4 == 2
Is not equal to	!=	4 != 2	4 != 4

These comparison operators are used in conditional statements. The conditional statements are used to control the flow of a program. The conditional statements are discussed below.

```
In [20]: # Is 2 less than 3?
         2 < 3

Out[20]: True

In [21]: # Is 2 greater than 3?
         2 > 3

Out[21]: False

In [23]: # Is 7 less than or equal to 7?
         7 <= 7

Out[23]: True

In [24]: # Is 7 greater than or equal to 7?
         7 >= 7

Out[24]: True

In [25]: # Is 3 equal to 3?
         3 == 3

Out[25]: True

In [26]: # Is 3 not equal to 3?
         3 != 3
```

```
Out[26]: False
```

Functions

Python functions are keywords in Python that have rules of use. The rules are called the **syntax** of the function. The syntax of a function is the rules that govern the use of the function. The `type` function that was used before is an example of a Python function. The syntax of the `type` function is `type(object)`. The `type` function takes a single argument, the object whose type is to be returned. The `type` function returns the data type of the object.

Python function will be covered later in these notes.

Python data types

Much of content that is used in data science in Python are of a certain computer data type. This type sets the rules for what can be done with the data. For example, a number can be added to another number, but a number cannot be added to text.

The `type` function comes in handy again. Below, are code examples of Python data types.

```
In [27]: # Type of 3
         type(3)
```

```
Out[27]: int
```

```
In [28]: # Type of 3.0
         type(3.0)
```

```
Out[28]: float
```

```
In [29]: # Type of 3.0 + 1j
         type(3.0 + 1j)
```

```
Out[29]: complex
```

```
In [30]: # Type of "Health Data Science"
         type("Health Data Science")
```

```
Out[30]: str
```

```
In [31]: # Type of '8'
         type('8')
```

```
Out[31]: str
```

The reason that a function has a type is that (almost) everything in Python is an object. To *function* properly, each object must have a type.

Casting or **coercion** converts an object from one type to another. Some casting is allowed in Python. The string `'8'` is coerced to an integer using the `int` function.

```
In [32]: # Convert the string '8' to an integer
         int('8')
```

```
Out[32]: 8
```

The string `'8'` can now be used in an arithmetic expression. The `float` function is demonstrated below in the same way as the `int` function. It coerces the string `'8'` to a floating point number 8.0.

```
In [33]: # Convert the string '8' to a float and add it to 3.0
         float('8') + 3.0
```

```
Out[33]: 11.0
```

Variables

Computer variables, or variables for short, are the containers for objects. Objects can be assigned to a variable. This action creates a space in computer memory in which the information about the object, including its type and value, are stored. At least while the Python program or file is active.

A *human-created name* is used as a variable name. These should be descriptive of what is contained in (or assigned to) the variable, such that when a file is viewed later, or shared with others, everyone can deduce what is contained in the variable.

Task: There are rules for naming a variable. These are included in a style-guide for writing Python code. The guides are listed in Python Enhancement Guide 8 (PEP 8), available at https://peps.python.org/pep-0008 or go directly to the Naming Convention section at https://peps.python.org/pep-0008/#naming-conventions. Read the Naming Convention section of PEP 8.

An object is assigned to a variable by using the assignment operator, which is the symbol `=`.

In the code cell below, a variable named `my_variable` is created and a string object is assigned to it. The string object contains the value `I like Python`.

```
In [34]: # Create a string object 'I like Python' and assign to the the variable my_v
         my_variable = 'I like Python'
```

Calling the variable name by simply typing the variable name in a code cell, will return the value of the object assigned to the variable.

```
In [35]: # Call the variable my_variable
         my_variable
```

```
Out[35]: 'I like Python'
```

Note that in a scripting environment such a notebooks, the `print` function,i.e. `print(my_variable)` , is not required to print the object in a variable to the screen.

The assignment operator is not used in the same way as it is used in mathematics. The assignment operator simply assigns what is to its right, to what is to its left. The variable name is on the left and the object is on the right.

Below, the computer variable `i` is created and the value 1 is assigned to it. The variable is then called.

```
In [36]: # Assign the value 1 to the variable i
         i = 1

         # Call the variable i
         i
```

```
Out[36]: 1
```

Now, 1 is added to the object contained in the variable `i` . The right-hand side is evaluated and the result is reassigned to the variable `i` . This means that the object assigned to a variable can be *over-written*. It is clear that the assignment operator is not used as an equal symbol.

```
In [37]: # Add 1 to i and assign the result to i
         i = i + 1

         # Call the variable i
         i
```

```
Out[37]: 2
```

On the right-hand side of the assignment operator is `i + 1` . When the code is excuted, `i` contains the value 1. When 1, is added to `i` the result is 2. This is then assigned to the variable that is on the left-hand side of the assignment operator. On the left is a variable that already contains an object. Python allows the content of a variable to be over-written. The variable `i` therefor then contains the integer object with a value of 2.

Task: Create code such that `x = 5` and `y = x`. Then let `x = 7`. Determine the value of `y`.

The control flow

The flow (or order of execution) of code can be controlled. This is done using conditional statements. To start with, though, consider the `range` function. The function is convenient for generating sequences. With a single-value integer argument, it generates a list starting at 0 and increments with a step size of 1 until the specified value, minus 1, is reached (the specified value is not included). The function returns a range object that can be used for iteration (over its elements).

In [38]:
```
# Use the range function to create a list of integers from 0 to 9
range(10)
```

Out[38]:
```
range(0, 10)
```

The `list` function prints the actual values.

In [43]:
```
# Use the list function to print the list of integers from 0 to 9
list(range(10))
```

Out[43]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

A `for` loop can be used to *loop over* the 10 values. In each case the current value is printed using the `print` function.

In [39]:
```
# Create a for loop to print the integers from 0 to 9
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

A starting value other than 0 can be specified. In the example below, the `range` function is used to generate a sequence starting at 1 and ending at 10 (not including 10).

In [40]:
```
# Use the range function to create a list of integers from 1 to 9
range(1, 10)
```

Out[40]:
```
range(1, 10)
```

The step-size can be controlled using a third argument.

In [44]:
```
# Use the range function to create a list of integers from 1 to 9 in steps o
list(range(1, 10, 2))
```

Out[44]:
```
[1, 3, 5, 7, 9]
```

Task: Generate a for loop that print the square of each integer from 0 to 5 to the screen. Remember to use the `**` symbols for calculating a power, and remember that the last value is not included.

A loop over code can also be achieved while a condition is met. A `while` loop will continue until a condition is met, or more precisely, the loop terminates when the condition returns a `False` value.

In [45]:
```
# Generate a variable i that takes the value 0
i = 0

# Create a while loop that prints the integers from 0 to 9
while i < 10: # Check if i is less than 10 (if it is, execute the code below
    print(i) # Print the current value of i
    i = i + 1 # Increment the current value of i by 1
```

```
0
1
2
3
4
5
6
7
8
9
```

Task: Initialize the variable `i` by assigning it the value 0. This overwrites the current value stored in `i` as used above. Print the square of each 1-unit increment in `i`, while `i` remains less than 10.

The `if`, `elif`, and `else` statements allow for branching of the execution of code. Depending on the set conditions, different code can be executed.

In the code cell below, a variable named `variable` is created, and assigned the integer value 10 to it.

In [46]:
```
# Create a variable named variable as assign to integer 10 to it
variable = 10
```

As an example of using an `if` statement, consider if the integer value in `variable` is larger than 5. If it is, print the statement `The value is larger than 5` to the screen. If the integer value in `variable` is less than or equal to 5, print `The value is not larger than 5` to the screen.

In [47]:
```
# Create an if-else expression to print 'The value is larger than 5' if the
if variable > 5:
    print('The value is larger than 5.')
else:
    print('The value is not larger than 5.')
```

```
The value is larger than 5
```

The keyword `if` is used and then a condition is stated, followed by a colon symbol, `:`. The new line is indented and contains the code for execution if the condition in the `if` statement holds (returns a `True` value). A new line uses the `else` keyword (in vertical allignment with the `if` keyword). The `else` keyword is also followed by a colon and then a new indented line containing a final statement to be executed if the prior condition was not met.

An `elif` (short for *else if*) statement can also be added. As an example, print `The value is exactly 5` if such a condition holds. The `elif` statememt (in vertical alignment with the `if` statement) is followed by a new conditional, a colon, and then a new indented line.

In [48]:
```
# Add an elif statement to the if-else expression above to print 'The value
if variable > 5:
    print('The value is larger than 5.')
elif variable == 5:
    print('The value is equal to 5.')
else:
    print('The value is not larger than 5.')
```

```
The value is larger than 5
```

In [49]:
```
# Change the value assigned to variable to 5
variable = 5

# Rerun the if-else expression above
if variable > 5:
    print('The value is larger than 5.')
elif variable == 5:
    print('The value is equal to 5.')
else:
    print('The value is not larger than 5.')
```

```
The value is equal to 5
```

It is a good idea think through the problem before creating an `if` statement.

Quiz questions

Questions

1. Use Python to calculate solutions to the following expressions.

$$3 \times 8$$

$$\frac{33 + 7}{5}$$

2. Determine the data type of the output of the code `1+2.` and `1+2`. Explain why the data types are different.

3. Generate a loop that prints n^2 for each of the numbers $n = \{1, 2, 3, 4, 5\}$.

4. Create the variable `i` by assigning the value 1 to it. Create a loop that prints the value of i^2 to the screen. Increment the value of `i` by 1 until the value of i^2 exceeds 1000. Print the value assigned to `i` after the loop has completed. Explain why the solution is `i=32`, where 32^2 is larger than 1000.

Solutions

1.

```
In [50]: 3 * 8
```

```
Out[50]: 24
```

```
In [51]: (33 + 7) / 5
```

```
Out[51]: 8.0
```

2. The code `1+2` returns an integer and the code `1+2.` returns a floating point number. The reason is that the decimal point in the second code is interpreted as a floating point number and the integer data type in Python cannot hold a floating point number.

3.

```
In [55]: for n in range(1, 6):
             print(n**2)
```

```
1
4
9
16
25
```

```
In [56]: [print(n**2) for n in range(1,6)]; # Alternative using list comprehension
```

```
1
4
9
16
25
```

4.

```
In [60]: i = 1

         while i**2 < 1000:
             i += 1

         print(i)
```

```
32
```

At `i=31` the value of `i**2` is 961 and at `i=32` the value of `i**2` is 1024. The value of `i` is 32 after the loop has completed.

```
In [ ]:
```