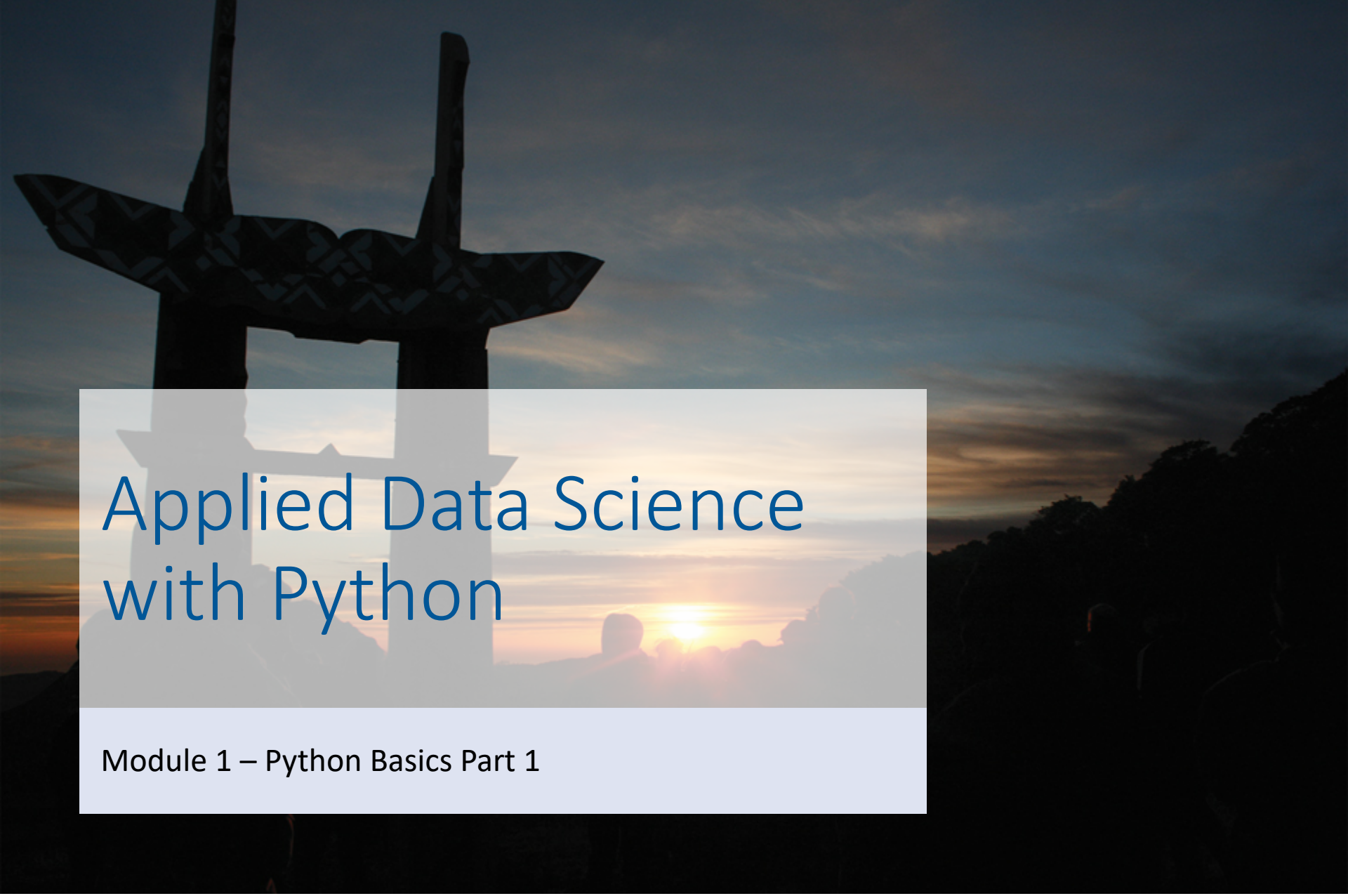


Applied Data Science with Python

Module 1 – Python Basics Part 1

Goal of the course

To cover the basics of Python

To learn how Python can be used for:

- Data Analysis
- Web Scraping
- Scientific Computing
- Plotting/Visualisation
- Machine Learning/Deep Learning

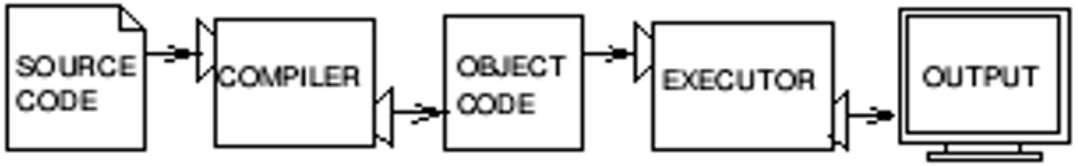
Why Python

- Very high Level Language
  - Provides many data structures like lists and dictionary
  - Shorter code compared to other languages
- Powerful libraries to perform complex tasks
  - Numpy, Pandas, Scipy, Scikit-learn, ...
- Cross platform, well documented and open source
- Object Oriented
- Extensible
- Widely adopted by the scientific community

How it works

- High Level Language ----->Interpreters

Low Level Language ----> Compilers



Python Installation

In this course we will use Python3.x and a few Libraries.

If you are familiar with a particular IDE(Integrated Development Environment) for Python you may go ahead and use this.

There are many IDE's that can be used, I would suggest maybe installing Anaconda, Pycharm or Visual Studio Code

For this course all our labs will be Jupyter Notebooks

Python Variables

A variable is a name that refers to a value, for example:

```
x = 1.51535

greeting = 'Hello World'

num = [1, 1, 2, 3, 5, 8]  # list
```

This character is use for comments

Legal Variable Names

Not all names can be used as a variable, there are some rules to remember.

- Names need to start with letters
- Only underscore can be used in special characters

Invalid examples include:

52cards = "Ace of Spades"

!cards = "Royal Flush"

Try this out:

class="How are you?"

Did this work?

Do the following to get a listing of Keywords that are reserved.

```
import keyword
keyword.kwlist
```

Types

Our variables can be various types.

For example if we type the following into a Jupyter Notebook using the type() function we would get the following outputs.

```
type("cars")
```

```
str
```

String of Character

```
type(42)
```

```
int
```

Integer

```
type(3.14)
```

```
float
```

Floating point number

Arithmetic Operations

5+12

12*4.514

(84-3+6)/5

'Hello' + ‘OTH'

‘spam’*3

String arithmetic operation: only **+** (also called concatenation) and ***** (for repeating) can be performed

Comparison Operators

Some of Pythons operators check whether a relationship holds between two objects.

Since the relationship either holds or doesn't hold, these operators are called Comparison Operators, and return a Boolean Value

Operator	Name	Example
==	Equal	A == B
!=	Not Equal	A != B
>	Greater than	A < B
<	Less than	A > B
>=	Greater than or equal to	A >= B
<=	Less than or equal to	A <= B

Boolean Operation

In programming you often need to know if an expression is **True** or **False.**

You can evaluate any expression in Python, and get one of two answers, **True** or **False**.

When you compare two values, the expression is evaluated and Python returns the Boolean answer. The bool() function allows you to evaluate any value.

Python uses **or, not, and** conveniently, instead of **||, !, &&.**

There is also the **in** operator that provides **True** or **False,** for example:

"e" in "this is my house"

Additionally, we can also chain our operators, for example:

2 < 3 < 4

Conditional Statements: if

Gives the ability to check the condition and change the behavior of the program accordingly.

```
x=-1
if x>0:
    print("x is positive")
if not(x>-1):
    print("x is negative")
if x==0:
    print("x is zero")
```

What is the expected results?

if/else

With if/else we are checking to see if a condition is **True**, else its **False**.

```
x = 3
if x%2 ==0:
    print("x is even")
else:
    print("x is odd")
```

Note: The **%** symbol refers to modulo, which refers to the remainder from a division operation.

While Loop

The while loop repeatedly executes a task while a condition is **True**.

```
number = 1
while number <=10:
    print(number)
    number +=1
print("Done!")
```

Break/Continue

Using **break** and **continue:**

- **Break** used to jump out of the loop
- Example: take user input until they type "**Done**"
- **Continue** is the opposite of **break**

for Loop

```
for (i = 0; i < n; i = i + 1)
```

Initialize index variable

Condition to Continue running the Loop

Increment at the end of each Loop

- Python uses ***for*** differently than Matlab, C++, ...
- ***for*** is used to iterate over elements in an “object”
- This is one reason why Python is easy and powerful

for Loop Example

Here are a few examples, Note the last example the third value –**1** is a step value.

```python
for x in range(0,3):
    print("the number is:",x)
```

```
the number is: 0
the number is: 1
the number is: 2
```

```python
for x in range(10):
    print("the number is:",x)
```

```
the number is: 0
the number is: 1
the number is: 2
the number is: 3
the number is: 4
the number is: 5
the number is: 6
the number is: 7
the number is: 8
the number is: 9
```

```python
for x in range(0,-10,-1):
    print("the number is:",x)
```

```
the number is: 0
the number is: -1
the number is: -2
the number is: -3
the number is: -4
the number is: -5
the number is: -6
the number is: -7
the number is: -8
the number is: -9
```

Strings

in Python, a string is a sequence of Unicode characters. Unicode was introduced to include every character in all languages and bring uniformity in encoding.

Strings in Python are surrounded by either single or double quotation mark and can be displayed with the print() function. For example:

print('Hello') or **print("Hello")**

```
1 "spam and eggs"
```

```
'spam and eggs'
```

```
1 hello = "greetings!"
```

```
1 hello
```

```
'greetings!'
```

```
1 print(hello)
```

```
greetings!
```

```
1 print(hello + " how are you?")
```

```
greetings! how are you?
```

Strings: len() function

The len() function can return the length of characters in a string

```
fruit = "banana"
```

```
length = len(fruit)
```

```
print(length)
```

```
6
```

```
print(fruit[length])  #expected error index out of range
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-31-959f9fea7b30> in <module>
----> 1 print(fruit[length])  #expected error index out of range

IndexError: string index out of range
```

```
print(fruit[-6])
```

```
b
```

Strings slices

- We can access individual characters using indexing and a range of characters using slicing. Index starts from 0.
- Trying to access a character out of index range will raise an IndexError.
- The index must be an integer.
- We can't use floats or other types, this will result into TypeError.
- Python allows negative indexing for its sequences.
- The index of -1 refers to the last item, -2 to the second last item and so on. We can access a range of items in a string by using the slicing operator :(colon).

```
s = "Monty Python"
```

```
print(s[0:5])
```

```
Monty
```

```
print(s[6:12])
```

```
Python
```

Strings are immutable

Cannot change existing String.

```
greeting ="hello, world"
greeting[0]='j' #error expected
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-72-822bcb3e69aa> in <module>
      1 greeting ="hello, world"
----> 2 greeting[0]='j'

TypeError: 'str' object does not support item assignment
```

```
new_greeting="J" + greeting[1:]
```

```
print(new_greeting)
```

```
Jello, world
```

Using the input() function

The input() function is used to take values from a user.

```
name= input("please enter your name?")
```

```
please enter your name?Istvan
```

```
print("hello", name)
```

```
hello Istvan
```

Functions

Named sequence of statements that performs a computation.

- Type conversion function.

```
1 int('32')
```

```
32
```

```
1 int("hello")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-36-a6f1987f81d0> in <module>
----> 1 int("hello")

ValueError: invalid literal for int() with base 10: 'hello'
```

Functions continued

- **math** function.
- Python has a "**math**" module for mathematical functions

```
import math
math.sqrt(2)/2
```

```
0.7071067811865476
```

Functions continued

- Your own functions
- Easier to read, test, fix, improve and develop
- Leads to more structured programming

```
def myAdd(value1, value2):
    total_sum=0
    total_sum= value1 + value2
    return total_sum

```

Get a user inputted value for the **x**.

```
x=int(input("provide a value for x: "))
```

```
provide a value for x: 5
```

Get a user inputted value for the **y**.

```
y=int(input("provide a value for y: "))
```

```
provide a value for y: 7
```

Call your function with your two inputted values

```
myAdd(x,y)
```

```
12
```