International Trade: Data Lab 1

Intro to Python

Carlos Góes[1]

[1]George Washington University

Fall 2025

Why Python?

Python for data science

- Python is a **general-purpose programming language** that is growing in popularity both in data science and other fields.
- Python prioritizes code readability for humans—and it is very easy for beginners.
- It follows principles such as (see the *Zen of Python*):
  - Explicit is better than implicit;
  - Simple is better than complex;
  - Readability counts.

Why Python?

Easy to understand

Java

```
class myprog
{
    public static void main(String args[])
    {
        System.out.println("Hi!");
    }
}
```

Python

```
print("Hi!")
```

Downloading Anaconda Navigator

Data science optimized environment

- Go on `https://www.anaconda.com/download`
- Skip registration
- Download your system's version

Introduction to Spyder

How to use Spyder?

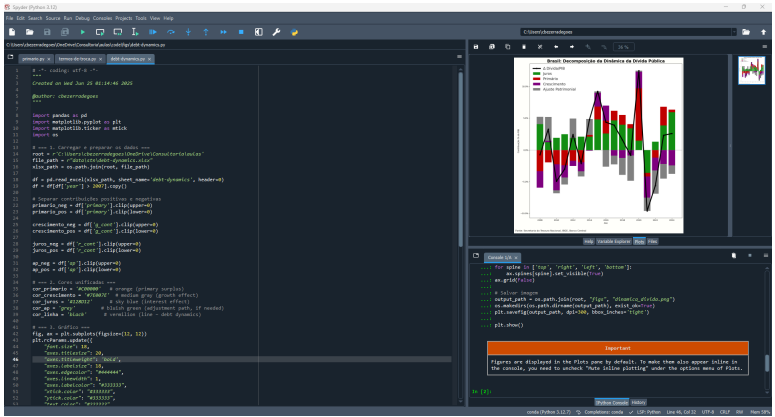

Figure: **Spyder.** Elements of Spyder's Integrated Development Environment (IDE).

Integers, floats, strings and booleans

Integers

- `Integers` are *whole numbers* (no decimals or fractions).

$$\mathbb{Z} \equiv \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\} \tag{1}$$

- Example in Python:

```
var = 20
print(type(var))
```

Integers, floats, strings and booleans

Floats

- `Floats` are arithmetic representations of all *real* numbers.

$$\begin{aligned} \mathbb{R} &\equiv \{\ldots, -\pi, -2.5, -\sqrt{2}, -\frac{1}{2}, 0, \frac{1}{2}, \sqrt{2}, 2.5, \pi, \ldots\} \\ \mathbb{R} &\equiv \{\ldots, -3.14159265, -2.5, -1.41421356, \\ &\quad -0.5, 0, 0.5, 1.41421356, 2.5, 3.14159265, \ldots\} \end{aligned} \tag{2}$$

- Example in Python:

```
var = 3.14159265
print(type(var))
```

Integers, floats, strings and booleans

Strings

- `Strings` are text representations.
- Example in Python:

```
var = "This is a string of characters"
print(type(var))
```

Integers, floats, strings and booleans

Booleans

- `Booleans` are logical representations (true or false).
- Example in Python:

```
var1 = True
print(type(var1))
var2 = (2+2 == 5)
print(type(var2))
```

Integers, floats, strings and booleans

Variable types and theoretical categorizations

Discrete quantitative variables: `integers`.
(e.g., completed years of schooling)

Continuous quantitative variables: `floats`.
(e.g., weight in lbs)

Categorical qualitative variables: `booleans` or `strings`.
(e.g., race)

Ordinal qualitative variables: `strings` or `integers`.

Integers, floats, strings and booleans

Numeric operations

- Addition:

```
x + y
```

- Subtraction:

```
x - y
```

- Division:

```
x / y
```

- Multiplication:

```
x * y
```

- Exponentiation:

```
x ** y
```

- Modulo (remainder):

```
x % y
3 % 2 is 1
```

Integers, floats, strings and booleans

String operations

- Addition equals concatenation:

```
str1 = "Carlos"
str2 = "Góes"
print(str1 + " " + str2)
```

- Multiplication equals repetition:

```
str1 = "a"
print(str1 * 10)
```

Integers, floats, strings and booleans

Logical operations

- Equality:

```
2 + 2 == 4
str = "a"
str == "a"
```

- Inequality:

```
2 + 2 != 4
str = "a"
str != "a"
```

- Greater than or less than:

```
10 > 100
10 < 100
```

Integers, floats, strings and booleans

Logical operations

- In logical operations, `True` has the numeric value 1 and `False` has the numeric value 0; you can use this to perform arithmetic.
- Sum (how many are true?):

```
a = (1 + 1) == 2
b = (2 * 2) == 4
c = (2 * 2) == 5
d = a + b + c
print(d)
```

- Product (are all true?):

```
a = (1 + 1) == 2
b = (2 * 2) == 4
c = (2 * 2) == 5
d = a * b * c
print(d)
```

Lists and dictionaries

Lists

- `Lists` are collections of variables defined with `[square brackets]` and elements separated by commas:

```
list1 = [2, 20.5, "Hi!", 10 < 100]
print(list1)
```

- `Lists` are indexed from 0 to the last element; access an element with the syntax `list_name[index]`.
- For example, `print(list1[0])` returns `2`.
- Meanwhile, `print(list1[3])` returns `True`.

Lists and dictionaries

Lists

- You can change a list element by assigning a value to a specific index:

```
list1[0]= 20
print(list1)
```

- Add an element to the list:

```
list1.append(79.2)
print(list1)
```

- Use multiplication to repeat the contents of a list:

```
list2 = list1 * 2
print(list2)
```

- Or use addition to concatenate two lists:

```
list3 = [1,2]
list4 = [5,6]
list5 = list3 + list4
print(list5)
```

Lists and dictionaries

Dictionaries

- `Dictionaries` are, as the name suggests, collections of key–value pairs; unlike lists, dictionaries are indexed by keys (usually words):

```
person1 = {
        'name': 'Milton Friedman',
        'dob': '07/31/1912'
        }
print(person1)
print(person1['name'])
```

- Add an element to the dictionary:

```
person1.update({'nationality': 'USA'})
print(person1)
```

Introduction to pandas

Importing the pandas

- The pandas package must be installed first!
- After that we can simply import it:

```
import pandas as pd
```

- pandas has two basic structures: Series and DataFrames; the latter are collections of the former.

Introduction to pandas

Series

- Series are built from other objects:

```
x = np.linspace(1,10, 5)
label = ["a","b","c","d","e"]
series1 = pd.Series(x, name="Series1", index=label)
print(series1)
```

- Like lists, you can access an element of a Series using its index:

```
print(series1["a"])
print(series1["d"])
```

Introduction to pandas

Series

- If you build your Series from dictionaries, they already come with indexes:

```
gwuid  = {
    'Carlos Goes': '06/99209',
    "Nicolas Powidayko": '10/22290',
    "Alexander Rabbat": '08/21346',
    "Dani Alaino": '07/20345',
    "Lya Nikate": '09/23567',
    "Niz Borroz": '11/22035',
    "Tom Rundal": "98/20145"
}

series2 = pd.Series(gwuid)
print(series2)
```

Introduction to pandas

DataFrames

- DataFrames are groups of Series:

```
x = np.linspace(1,10, 5)
y = np.linspace(1,20, 5)
label = ["a","b","c","d","e"]
series1 = pd.Series(x, name="Series1", index=label)
series2 = pd.Series(y, name="Series2", index=label)

df = pd.DataFrame(data=[series1, series2])
```

- You can extract both columns and rows:

```
print(df["a"])
print(df.loc["Series1"])
```

- And transpose (flip) the data:

```
print(df.T)
```

Introduction to pandas

DataFrames

- You can also grab a specific element inside a column: `print(df["a"]["Series1"])`
- Or call the row first and then the column: `print(df.loc["Series1"]["a"])`

Introduction to pandas

DataFrames

- Let's create a DataFrame with several attributes:

```
gwuid = pd.Series(gwuid)

major = pd.Series({
        'Carlos Goes': 'Economics',
        "Nicolas Powidayko": 'Economics',
        "Alexander Rabbat": 'Computer Science',
        "Dani Alaino": 'Computer Science',
        "Lya Nikate": 'Computer Science',
        "Niz Borroz": 'Statistics',
        "Tom Rundal": "Computer Science"
        })

gpa = pd.Series({
        'Carlos Goes': 4.0,
        "Nicolas Powidayko": 3.8,
        "Alexander Rabbat": 3.8,
        "Dani Alaino": 3.4,
        "Lya Nikate": 3.3,
        "Niz Borroz": 3.0,
        "Tom Rundal": 3.0
        })

list = [gwuid, major, gpa]
df = pd.DataFrame(list, index=['matricula', 'major', 'ira']).T
```

Introduction to pandas

DataFrames

- How to extract the attributes of Carlos Goes? `df.loc["Carlos Goes"]`
- How to extract all registration numbers? `df["gwuid"]`
- How to extract the data for all Computer Science students?
  - *Boolean masking*!
  - Try: `print(df["major"] == "Computer Science")`
  - And now like this: `print(df[ df["major"] == "Computer Science" ])`
  - What happened?