Python Crash Course

Prerrequisites

- Knowledge of some programming language (not necessarily Python).
- Download and install Python from the official website.
- Download and install Visual Studio Code from here.

What is Python?

- Interpreted, dynamically typed language.
- Used as a "glue" language (combining different systems) and as a standalone language:
  - Data analysis / machine learning / scientific computing.
  - Web development (Django, Flask, FastAPI).
  - System Administration.
  - GUI.
- Relatively easy to learn, with concise syntax.

Executing Python files

- Write a file with extension `.py` and then execute `python myfile.py`.
- Use the interactive interpreter in the command line.
- Use Jupyter Notebooks (web-based IDE).

Examples

hello.py

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

hello.py

```python
print("Ahoj!")
```

- `print` is a **function**.
- `"Ahoj"` is an **argument**.

name.py

```python
name = input()
print(f"Hi {name}!")
```

Data Types

- Python does not require you to specify it from the beginning (unlike for instance C++ or Java).
    - Supported types: Integer, Float, Boolean, None.
- You can check the type with `type()` function.

Conditional Statements

```
x = input()

if x>0:
        print("x is positive")
elif x<0:
        print("x is negative")
else:
        print("x is zero")
```

Sequences

```python
name = "Pablo"
records = (123, "Some User", 721324890, 123.45)
names = ["Pablo", "Marek", "Petra", "Jana"]
letters = 'abcdefg'
```

Indexing

- First element in a sequence is `0`.
- Last element is `-1`.
- `letters[1:7:2]` returns `'bdf'`.

Strings

- There is no "character" type in Python.
- `'Pablo'.upper()`
- `'Pablo'.lower()`
- `nosferatu`.title()
- `'P' in 'Pablo'`
- `'Pablo;Maldonado;Prague'.split(';')`
- `'_'.join(['Pablo', 'Maldonado'])`

Lists

- `names = ["Pablo", "Marek", "Petra", "Jana"]`
- `names.append("Olga")`
- `names.append("Elmo")`
- `names.pop()`
- `names.remove("Marek")`
- `sorted(names) # returns sorted list`
- `names.sort() # in-place sorting`
- `names[1] = 'Pavel'`

Tuples

- Tuples are similar to lists, except that they are immutable.

```
records = (123, "Some User", 721324890, 123.45)
records[1] = "Another User" # will fail
```

Dictionaries

- Collections of key-value pairs.

```
ages = {"Pablo":25, "Olga":30}
ages["Pablo"] = 30
ages["Olga"] += 1
```

- Dictionaries can nest other values

```
user1 = {'name':'Pablo', 'hobbies':['box','movies','beer']}
user2 = {'name':'Petra', 'hobbies':['climbing','beer']}
users = {'u-001':user1, 'u-002':user2}
```

Loops

- Simple loops.

```python
for i in range(5):
        print(i)
```

- Loops over sequences.

```python
names = ["Pablo", "Marek", "Petra", "Jana"]
for name in names:
        print(name)
```

More Loops

- Loops over dictionaries

```python
ages = {"Pablo":25, "Olga":30}
ages["Pablo"] = 30
ages["Olga"] += 1

for key, value in ages.items():
        print(f'The age of {key} is {value}')
```

- Conditions on loops.

```python
i = 0
while i < 10:
        print(i)
        i += 1 # i = i+1
```

Functions

```python
def square(x):
   return x*x

for i in range(10):
   print("{} squared is {}".format(i, square(i)))
```

Exercises

- Calculate the number of vowels in a string.
- Calculate the number of capital letters in a string.
- Consider a `beers` dictionary defined as follows:

```
beers = {
    'Kozel':30,
    'Pilsner':40,
    'Matuska':60,
    'Antos':55
    }
```

Write a program that takes a user input of the form: `beer1;beer2;beer3` and calculates their bill. You can use the `sum(<list>)` function, where `<list>` represents a list of values.

List comprehensions

- Short notation to apply operations on lists.

```
numbers = (1, 2, 3, 4, 5)
squares = [num**2 for num in numbers]
```

```
vals = ["1.5", "3.14", "9.87"]
float_vals = [float(val) for val in vals]
```

List comprehensions (cont.)

```python
vals = ["1.5", "3.14", "9.87"]
vals = [float(val) for val in vals if float(val) < 5]
vals_mask = [1 if float(val) < 5 else 0 for val in vals]
```

List comprehensions and lambda functions.

```
parse = lambda val: float(val)
vals = [parse(val) for val in vals]
```

Exercise

- Given a string of values separated by a whitespace, that is, something like this: <center><code>10 abc 20 de44 30 55fg 40</code> </center> write a function that identifies the numbers and sums them. In this case, the output should be 100.

Modules and packages

- You can invoke pieces of code from one file into another.

functions.py

```
def square(x):
        return x*x

for i in range(10):
        print("{} squared is {}".format(i, square(i))
```

modules.py

```
from functions import square
print(square(10))
```

- **Trivia:** What would happen when you run `modules.py`?

Modules and packages (cont.)

- When you import a file, Python runs **everything** inside by default. We need to modify functions.py to prevent this.

```
def square(x):
        return x*x

def main():
  for i in range(10):
      print("{} squared is {}".format(i, square(i))

if __name__ == "__main__":
        main()
```

Modules and packages (cont.)

- `if __name__ == "__main__"` simply means: *execute the code below only if this is the main program being run.*
- To import a file from a subfolder, you need to create an empty `__init__.py` file in the folder first.

Classes

```
import math
class Point:
        def __init__(self, x, y):
        self.x = x
        self.y = y

        def magnitude(self):
        return math.sqrt(x*x+y*y)

p = Point(3,5)
print(p.x)
print(p.magnitude())
```

Classes and Inheritance

```
class Car:
    def __init__(self, brand, model, year):
        self.brand = brand
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def __str__(self):
        txt = f"A {self.year} {self.brand} {self.model}"
        return txt
```

Classes and Inheritance (cont.)

```
class ElectricCar(Car):
    def __init__(self, brand, model, year, battery_life):
        super().__init__(brand, model, year)
        self.battery_life = battery_life

    def __str__(self):
        base = "A {0} that goes for {1} km"
        txt = base.format(self.brand,self.battery_life)
        return txt
```

Decorators

```
def argument_test_natural_number(f):
    def helper(x):
        if type(x) == int and x > 0:
            return f(x)
        else:
            raise Exception("Argument is not an integer")
    return helper

@argument_test_natural_number
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)
for i in range(1,10):
        print(i, factorial(i))
print(factorial(-1))
```

Handling files

- `open(filename, mode)`
- Mode can be one of the following:
  - "r" - Read - Default value. Opens a file for reading, error if the file does not exist
  - "a" - Append - Opens a file for appending, creates the file if it does not exist
  - "w" - Write - Opens a file for writing, creates the file if it does not exist
  - "x" - Create - Creates the specified file, returns an error if the file exists

In addition you can specify how the file should be handled:

- "t" - Text - Default value. Text mode
- "b" - Binary - Binary mode (e.g. images)

Handling files (cont.)

- `f = open("demofile.txt")`
- `f = open("demofile.txt", "rt")`
- Files **must** be closed manually in this syntax.

Handling files (cont.)

- It is often preferred to use **context managers** for handling files:

```
with open("demofile.txt") as f:
    f.read()    # f.write()
```

Exceptions

```
try:
  print(x)
except:
  print("An exception occurred")
```

Exceptions (cont.)

```
try:
  f = open("demofile.txt")
  try:
    f.write("Lorum Ipsum")
  except:
    print("Something went wrong when writing to the file")
  finally:
    f.close()
except:
  print("Something went wrong when opening the file")
```

Exceptions (cont.)

```
x = "hello"

if not type(x) is int:
  raise TypeError("Only integers are allowed")
```

Case Study

Solution

```
import csv
import glob


def validate(line):
    return len(line.split(',')) == 6 # should be 6 records

def parse(line):
    row = line.split(',')
    new_line = None

    if row[0] == 'SBN':
        new_line = row

    elif int(row[1]) >= 17: # Exclude sensor measures 17+
        return None

    else:
        new_line = [int(r, 16) for r in row]

    return ','.join([str(it) for it in new_line])+'\n'
```

Solution (cont.)

```
log_files = glob.glob('./data/logs/*')
out_file = './data/clean_file.csv'

with open(out_file, mode='w') as outfile:
    for log_file in log_files:
        with open(log_file, mode='r') as logdata:
            for line in logdata:
                if validate(line):
                    new_line = parse(line)
                    #save(new_line)
                    if new_line is not None:
                        outfile.write(new_line)
```

Exploring data with pandas

```
import pandas as pd
df = pd.read_csv("./data/clean_file.csv")
df.head()
df.info()
```

Exploring data with pandas (cont.)

- Filter rows fulfilling a certain condition

```
df[(df['SBN']==1) & (df['CN'] == 1) ]
device = df.query('SBN==1 & CN==1')
```

Exploring data with pandas (cont.)

```
device.index = pd.to_datetime(device['timestamp'], format='%Y-%m-%d-%H-%M-%S')
device['T'].plot();
device[['x', 'T']].plot(ylim=(45000,55000), title='Plot of x vs T');
```

Exploring data with pandas (cont.)

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(nrows=1, ncols=2, figsize = (14,5))
device['T'].plot(ax=ax[0], title='Plot of T', color='red', linestyle='dashed')
device['x'].plot(ax=ax[1], title='Plot of x')
```

Exploring data with pandas (cont.)

```
fig, ax = plt.subplots(nrows=2, ncols=1, sharex=True, figsize = (14,5))
device['y'].plot(ax=ax[0], title='Plot of y', color='red', linestyle='dashed')
device['x'].plot(ax=ax[1], title='Plot of x')
```

Exploring data with pandas (cont.)

```python
import seaborn as sns
sns.heatmap(device[['x','y','z','T']].corr(), annot=True, center=0, cmap='RdBu')
sns.jointplot(x="x", y="T", data=device)
```

Useful References

- First 39 minutes of this video.
- Python for Everybody