George Kantasis

[Data Science - Week 01]

4Nov24

TechPro

Introduction

- Setting up Python and Integrated Development Environment (IDE)
- Basic syntax of a Python program
- Variables and Data Types
- Operators
- Basic input and output

What is Python

- A general-purpose language
- Uses:
  - AI, Deep Learning, Machine Learning
  - Web development
  - IoT

What is Python

Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

Python is Interactive: You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Dynamically Typed: Python is a dynamically typed programming language. In Python, you don’t need to specify the variable time at the time of the variable declaration. The types are specified at the runtime based on the assigned value due to its dynamically typed feature.

Why Python

Unlike other powerful languages, Python does not require the programmer to have deep knowledge of computer science, compiler quirks and operating system temperaments to effectively use it.

This allows users from diverse backgrounds to dive into their fields of study more easily

- Mathematics
- Physics
- Statistics
- Demand forecasting
- Automation
- FinTech
- Cybersecurity

Zen of Python

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated

Python IDE

- Notepad
- Notepad++
- VSCode
- Notebooks
  - Google Colab
  - Jupyter
  - Kaggle Notebooks
- Pycharm
- Sublime

Instalation (linux)

```
sudo apt-get update -y
sudo apt-get install python3, python3-pip -y
python3 --version
```

Instalation (windows)

Download the installer from

https://www.python.org/ftp/python/3.11.2/python-3.11.2-amd64.exe

And follow the wizard

Instalation (mac)

Get out.

PATH

If the python command is not recognised, check if the PATH variable contains the path to the executable

```
# In bash
# If this command does not give a string with a valid python path...
echo $PATH
# execute this command. It will work until you close the console
export PATH="$PATH:/usr/local/bin/python"
```

Python IDE

- **-d**: It provides debug output.
- **-O**: It generates optimized bytecode (resulting in .pyo files).
- **-S**: Do not run import site to look for Python paths on startup.
- **-v**: verbose output (detailed trace on import statements).
- **-c cmd**: run Python snippet
- **file**: run Python script from given file
- **--version**

Hello World:

Unlike most languages where the simplest program is quite verbose;

```c
#include <stdio.h>

int main() {
   printf("Hello, World!\n");
   return 0;
}
```

```java
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello, World!");
   }
}
```

TechPro

Hello World:

In Python, the basic program is simple and terse

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

Interactive Mode

Open a python console and start typing commands

A console is the most basic and bare-bones interface with a computer

You use plain-text to write commands and the programm responds with plain-text

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

Try it out!

Scripts

- Other than that, you can create a **text file** with the code you want to execute and ask python to read this file.
- Of course this is not interactive.
- For instance consider a file named `test.py`.
- Conventionally we use the `*.py` suffix for python code files

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

- And run it like so

```
> python3 test.py
```

Scripts 2: The shebang

Perhaps invoking the `python3` command is too much of a hassle.

```
> python3 test.py
```

We want to run it like a real program, like:

```
./test.py
```

In that case we should use a shebang in the source file

```
#!/usr/bin/python3

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

Scripts 3: The shebang

Does it work?

In linux systems you may need an additional (secret) step

```
> chmod +x test.py
```

This tells the operating system that this text file is not just to be read and written to. It can also be executed.

It’s a safety precaution

Variables

In Python, variables are created when you assign a value to it:

```
# Assign a variable
x = 2

# Print the variable
print(x)

# Print an expression
print(x+1)

# Change the value of the variable
x=x+5
x=x*2
x="lorem ipsum"

# What will this print?
print(x)
```

TechPro

Variable names

- Variable names are case-sensitive
- They can't be keywords (like `if` or `print`)
- They can only contain alphanumeric or underscore characters
- They can't begin with a number

Variable names 2: Conventions

- **Camel Case:** When using multi-word variables, you can capitalize each first letter but the first: `myVariableName`
- **Pascal Case:** Like camel case, but you also capitalize the first letter: `MyVariableName`
- **Snake Case:** Each word in the variable name is separated with an underscore: `my_variable_name`

Conventions:

- We use camel case for general variables

- We use pascal case for class names

- We prepend underscore for variables to denote them as 'private'

Data Types

`**str**`: String

`**int**`: Integer

`**float**`: Numbers with decimals

`**complex**`: Real and Imaginary

`**list**`: a list of values

`**tuple**`: an immutable list of values

`**set**`: a list of values with no duplicates

`**dict**`: a list of key-value pairs

`**bool**`: yes/no

`**None**`: nothing

Data Types 2: Casting

We can change the data type of a variable using a casting function

```
x = str(3)
x = int(3.0)
x = float(3)
x = int(“5”)
```

What do you think the above commands will result to?

TechPro

Data Types 3: type()

the `type()` function returns the data type of its argument

```
x = 5
y = "John"
print(type(x))
print(type(y))
```

Try this out in the interactive console

Data Types 4: Examples

What data types are the following assignments?

```
x = "Hello World"
x = 20
x = 20.0
x = 1j
x = 12E4
x = ["apple", "banana", "cherry"]
x = ("apple", "banana", "cherry")
x = {"name" : "John", "age" : 36}
x = {"apple", "banana", "cherry"}
x = True
x = False
x = None

x = int(20.0)
x = float(20.0)
x = str(20.0)
x = dict(name="John", age=36)
x = list(("apple", "banana", "cherry"))
x = tuple(("apple", "banana", "cherry"))
```

TechPro

User input

Just as `print()` is a python function that outputs data to the console, the `input()` function is used to input data from the console to the program.

```
name = input("Enter your name: ")
print("Hello, " + name + "!")
```

The `input()` function stops execution of the programm (that's called "blocking") and prints a message.

The user then should type a string and press enter.

The function then assigns this string to the variable "name"

Wait, what does `"Hello, " + name + "!"` mean?

Strings

Strings are arrays of characters, letters, numbers etc

It’s a data type that holds text data

```
# In python we denote strings as text between single or double quotes
x="Hello"
x='Hello'

# The text can also have quotes
x="My mom said \"yes\". So we can go."

# Strings can span multiple lines
x="""
My mom said "yes".
So we can 'go'.
"""
print(x)
```

Strings 2

Strings also provide some additional utilities to handle the data

Making our lives easier

```
# We can also concatenate strings
x="Hello"+"World"

# f-strings
x="George"
y=f"My name is {x}"
```

Strings 3: Escape Characters

There are some characters that require an additional step to be represented in a string

For example we've seen that you need a backslash (\) to represent a quote(')

What if I want to represent a backslash?

Character	Name	Code
'	Single Quote	\'
"	Double Quote	\"
\	Backslash	\\
	New Line	\n
	Carriage Return	\r
	Tab	\t
	Backspace	\b

Operators 1: Arithmetic Operators

- **+:** addition
- **-:** Subtraction
- **/:** division
- **//:** integer division
- ***:** multiplication
- ****:** power
- **%:** remainder

Operators 2: Logic Operators

- **and**: Logic AND
- **or**: Logic OR
- **not**: Logic NOT

Operators 3: Comparison Operators

- **<:** Less than
- **>:** Greater than
- **==:** Equals (attention here)
- **!=:** Not equals
- **<=:** Less than or equal
- **>=:** Greater than or equal

Operators 4: Other Operators

- **is:** Identity Operator, returns True if both expressions are the same object
- **is not:** negation of the above
- **in:** Membership Operator, returns true if an element is present in the object
- **not in:** negation of the above

Order of operations

What is the result of 6/2*(1+2)?

The order that you perform the operations matters

In any expression, Python evaluates the operations from right to left in the following order:

- (): Inner Parentheses
- (): Outer Parentheses
- **: Powers
- *, /, //, %: Multiplicative operations
- +, -: Addition and subtraction
- <,>,==,<=,>=,!=: Logic comparisons
- not: Logic not
- and: Logic AND
- or: Logic OR

Order of operations 2

However, when writing code remember:

When in doubt

Use parentheses

Even nested ones, it's perfectly fine.

Boolean

Booleans represent one of two values: True or False.

We call expressions that evaluate to a boolean, conditions

Expressions can be evaluated to a boolean value.

For instance the number 10 can be evaluated to True.

Most values on their own are True

Exception 1: An empty string/list/set/dictionary

Exception 2: An arithmetic expression that evaluates to 0

Exception 3: The None value

`if` statements

This should be quite self-explanatory

```
if 5 > 2:
  print("Five is greater than two!")
```

To declare an `if` statement, you need to

- use the `**if**` keyword,
- follow it by a **condition**
- and close up with a colon `:`

The **indented lines** that immediately follow will be executed **only if** the condition is **true**

`if` statements 2: The `else`

It’s that easy

```
if 5 > 2:
   print("Five is greater than two!")
else:
   print("Five is not greater than two")
```

The optional `else` statement executes the **indented lines** immediately after it if the condition is **false**

`if` statements 3: the `elif`

`elif` statements stand for "else if" statements and check for an additional condition if the original was false

```
if 5 > 2:
  print("Five is greater than two!")
elif 5<2:
  print("Five is less than two")
```

you can chain `elif` statements together:

```
num=1000
if num<10:
   print(f"{num} is a single-digit number")
elif num<100:
   print(f"{num} is a two-digit number")
elif num<1000:
   print(f"{num} is a three-digit number")
else:
   print(f"{num} is big af")
```

`if` statements 4: The ternary operator

```
age = 18

# The following is the regular way of approaching the code
if age >= 18:
   status="Adult"
else:
   status="Minor"
print(status)

# But if your code is super simple like that,
# you could also use the following syntax:
status = "Adult" if age >= 18 else "Minor"
print(status)
```

Indentation Exercise

```
# Is this Correct?
print("hello world")
print("hello world")

# Is this Correct?
print("hello world")
  print("hello world")

# Is this Correct?
if 5 > 2:
  print("Five is greater than two!")
  print("Five is greater than two!")
```

Indentation Exercise 2

```
# Is this Correct?
if 5 > 2:
     print("Five is greater than two!")

# Is this Correct?
if 5 > 2:
  print("Five is greater than two!")
print("Five is greater than two!")

# Is this Correct?
if 5 > 2:
  print("Five is greater than two!")
     print("Five is greater than two!")
```

Indentation Exercise 3

```
# Is this Correct?
if 5 > 2:
     print("Five is greater than two!")
  print("Five is greater than two!")

# Is this Correct?
if 5 > 2:
  print("Five is greater than two!")
else:
  print("Five is greater than two!")

# Is this Correct?
if 5 > 2:
  print("Five is greater than two!")
else:
     print("Five is greater than two!")
```

Indentation Exercise 4

```
# Is this Correct?
if 5 > 2:
  print("Five is greater than two!")
else:

# Is this Correct?
if 5 > 2:
else:
  print("Five is greater than two!")
```

Recap

What did we see today?

Can you write a python script?

Summary

- A brief introduction to Python
- Python Installation and IDE
- Basic programming in Python
- Syntax and code structure
- Variables, Data Types and Operators
- The `if` statement