(Monty) Python

and Google Colab

by **Martin Schätz**

Motivation

& what to learn

- What is Python
- The easiest way to use it
- The basics
- Use magic
- Create stunning charts
- Work with huge tables
- Do statistics

Why is it called Python?

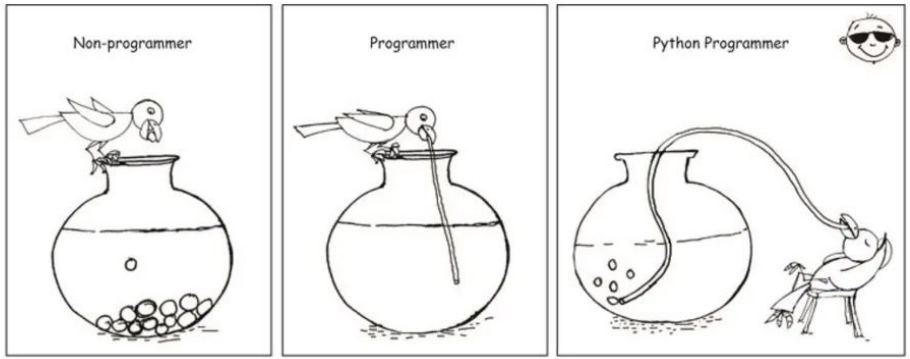
When he began implementing Python, Guido van Rossum was also reading the published scripts from "***Monty Python's Flying Circus***", a BBC comedy series from the 1970s. Van Rossum thought he needed a name that was short, unique, and slightly mysterious, so he decided to call the language Python.

If you read the Python documentation, you will see many examples that are inspired by the Monty Python comedy series:

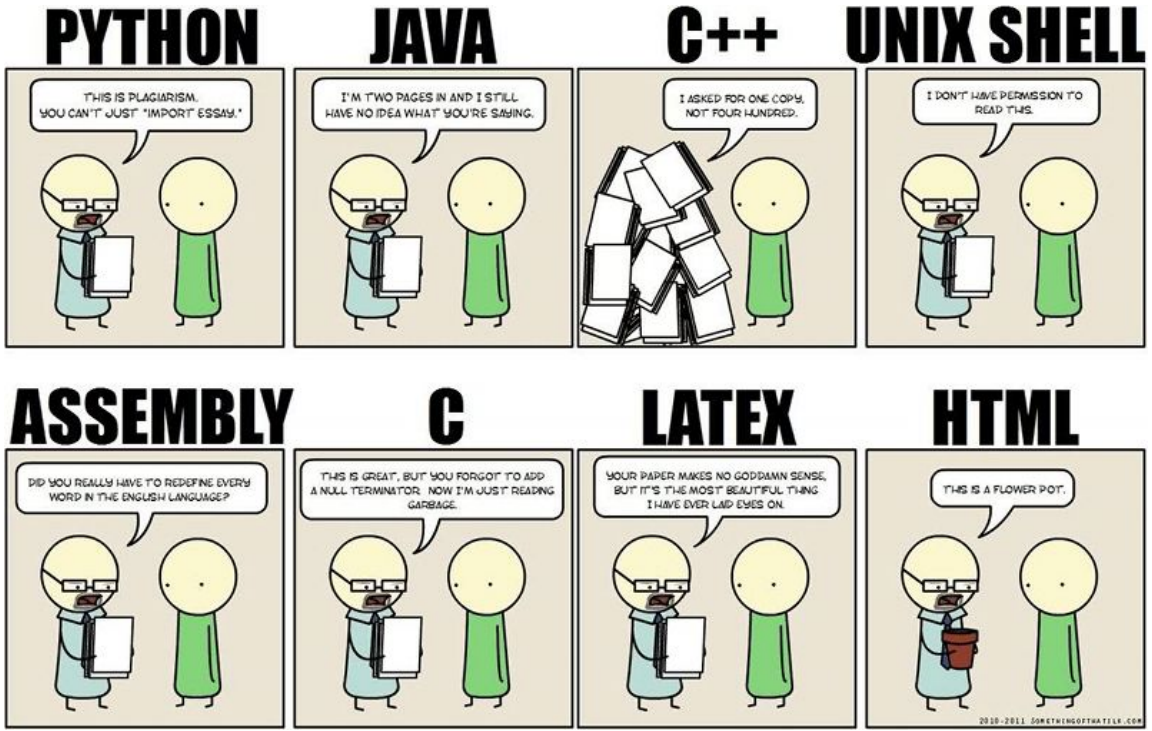
You will see mention of spam, eggs, lumberjack, and knights, etc.

Why to use Python?

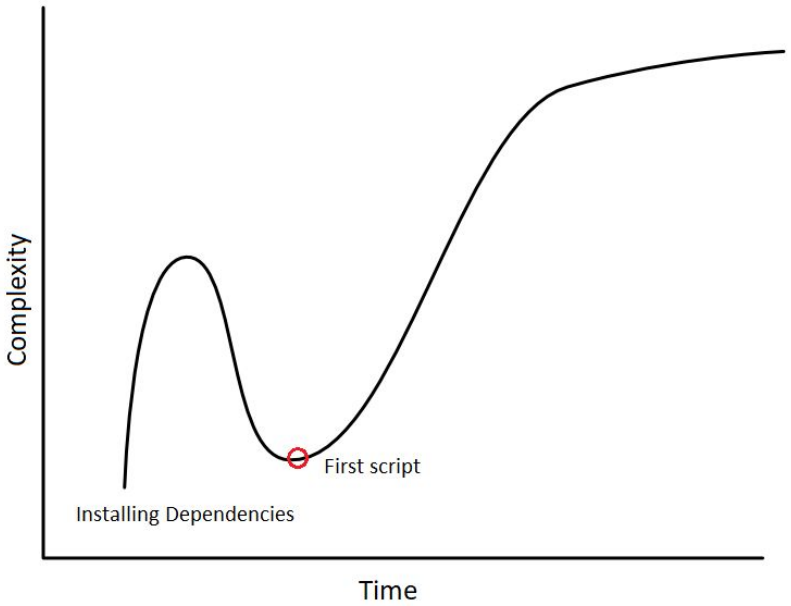
1. Python is easy to read, write, and learn
2. Python is extremely versatile, with multiple uses
3. Python has an incredibly supportive community
4. Python is the fastest growing programming language
5. Python is less complex for Data Science and Machine Learning



But... is it really that easy?



But... is it really that easy?



Google Colaboratory

Colab is a free Jupyter notebook environment that runs entirely in the cloud. Most importantly, it does not require a setup and the notebooks that you create can be simultaneously edited by your team members - just the way you edit documents in Google Docs.

What Colab Offers You?

- Write and execute code in Python
- Document your code that supports mathematical equations
- Create/Upload/Share notebooks
- Import/Save notebooks from/to Google Drive
- Integrate PyTorch, TensorFlow, Keras, OpenCV
- Free Cloud service with free GPU

Python - basics for scripting

Comments, text and numbers

```
# this is the first comment
spam = 1  # and this is the second comment
          # ... and now a third!
text = "# This is not a comment because it's inside quotes."
```

```
>>> 2 + 2
4

>>> 50 - 5*6
20

>>> (50 - 5*6) / 4
5.0

>>> 8 / 5  # division always returns a floating point number
1.6
```

Numbers and operators

```
>>> 17 / 3  # classic division returns a float
5.666666666666667

>>> 17 // 3  # floor division discards the fractional part
5

>>> 17 % 3  # the % operator returns the remainder of the division
2

>>> 5 * 3 + 2  # floored quotient * divisor + remainder
17

>>> 5 ** 2  # 5 squared
25

>>> 2 ** 7  # 2 to the power of 7
128
```

Strings

```
>>> 'spam eggs'  # single quotes
'spam eggs'
>>> 'doesn\'t'  # use \' to escape the single quote…
"doesn't"
>>> "doesn't"  # ...or use double quotes instead
"doesn't"
>>> '"Yes," they said.'
'"Yes," they said.'
>>> "\"Yes,\" they said."
'"Yes," they said.'
>>> '"Isn\'t," they said.'
'"Isn\'t," they said.'
```

```
>>> word = 'Python'
>>> word[0]  # character in position 0
'P'
>>> word[5]  # character in position 5
'n'
>>> word[-1]  # last character
'n'
>>> word[-2]  # second-last character
'o'
>>> word[-6]
'P'

>>> word[0:2]  # characters from position 0 (included) to 2 (excluded)
'Py'
>>> word[2:5]  # characters from position 2 (included) to 5 (excluded)
'tho'
```

Lists

```
>>> squares = [1, 4, 9, 16, 25]
>>> squares
[1, 4, 9, 16, 25]
>>> squares[0]  # indexing returns the item
1
>>> squares[-1]
25
>>> squares[-3:]  # slicing returns a new list
[9, 16, 25]

>>> squares + [36, 49, 64, 81, 100]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

 squares.append(216)  # add 216
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 216]
```

Dictionaries

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'jack': 4098, 'sape': 4139, 'guido': 4127}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'jack': 4098, 'guido': 4127, 'irv': 4127}
>>> list(tel)
['jack', 'guido', 'irv']
>>> sorted(tel)
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
>>> 'jack' not in tel
False
```

```
>>> dict([('sape', 4139), ('guido', 4127),
('jack', 4098)])
{'sape': 4139, 'guido': 4127, 'jack': 4098}

>> {x: x**2 for x in (2, 4, 6)}
{2: 4, 4: 16, 6: 36}

>>> dict(sape=4139, guido=4127, jack=4098)
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

Flow Control - IF

```
>>> x = int(input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...     x = 0
...     print('Negative changed to zero')
... elif x == 0:
...     print('Zero')
... elif x == 1:
...     print('Single')
... else:
...     print('More')
…
More
```

Flow Control - WHILE

```
>>> # Fibonacci series:
... # the sum of two elements defines the next
... a, b = 0, 1
>>> while a < 10:
...     print(a)
...     a, b = b, a+b
…
0
1
1
2
3
5
8
```

Flow Control - FOR

```
>>> # Measure some strings:
... words = ['cat', 'window', 'defenestrate']
>>> for w in words:
...     print(w, len(w))
…
cat 3
window 6
defenestrate 12
```

The *range()* Function¶

```
>>> for i in range(5):
...     print(i)
…
0
1
2
3
4
```

```
range(start, stop, step)
```

```
range(0, 10, 3)
   0, 3, 6, 9
```

Flow Control - FOR + BREAK and CONTINUE

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(n, 'equals', x, '*', n//x)
...             break
...     else:
...         # loop fell through without finding a factor
...         print(n, 'is a prime number')
…
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

Flow Control - FOR + BREAK and CONTINUE

```
>>> for num in range(2, 10):
...     if num % 2 == 0:
...         print("Found an even number", num)
...         continue
...     print("Found an odd number", num)
…
Found an even number 2
Found an odd number 3
Found an even number 4
Found an odd number 5
Found an even number 6
Found an odd number 7
Found an even number 8
Found an odd number 9
```

Defining function

```
>>> def my_function():
...     """Do nothing, but document it.
…
...     No, really, it doesn't do anything.
...     """
...     pass
...

>>> print(my_function.__doc__)
Do nothing, but document it.
    No, really, it doesn't do anything.
```

Defining function

```
>>> def f(ham: str, eggs: str = 'eggs') -> str:
...     print("Annotations:", f.__annotations__)
...     print("Arguments:", ham, eggs)
...     return ham + ' and ' + eggs
…
>>> f('spam')
Annotations: {'ham': <class 'str'>, 'return': <class
'str'>, 'eggs': <class 'str'>}
Arguments: spam eggs
'spam and eggs'
```

*Monty Python & the origin of word spam