Python – Crash Course

Learning the python essentials

Why Python over Java/C/C++/everything else?

Advantages

- Third most popular language after Java / C
- Easy to learn and use
- Multitude of libraries
- Ships with most Linux distros
- Interpreted, not compiled
- Less lines of code (compared to Java/C++)
- Dynamic typing
- No semicolons or curly brackets (kind of)

Disadvantages

- Slower than compiled languages
- Higher level = less control than e.g. C/C++
- Higher memory usage
- Not good for mobile development

Real world examples of Python

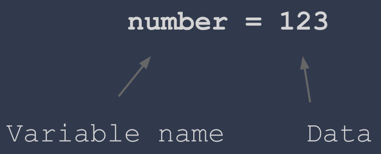
- **Instagram backend** - “Instagram Server is entirely **Python** powered.” (Django Framework)
- **Reddit** - “Reddit was originally written in Common Lisp but was rewritten in **Python** in December 2005”
- **Eve Online (MMORPG)** - “Both the server and the client software for Eve Online are developed in Stackless **Python**, a variant of the **Python** programming language.”
- **Dropbox** - “Dropbox has about four million lines of **Python** code and it's the most heavily used language for its back-end services and desktop app” (Also where the creator of the language worked 2013-2019)
- **SMHI** - “Its **Python**-based remote sensing software for automatic product generation, using NOAA and Meteosat data, provides information to bench forecasters, objective analysis schemes, and commercial interests such as the media
- And **MANY** more!

QUICK rundown of python essentials

- **Variables** (data containers)
- **Flow control** is handled with:
  - **If, elif and else conditionals**
  - **For, while loops**
- **Functions** are blocks of code that can be executed at will and takes parameters (arguments) and usually returns a result, for example the user input function **input()** our output function **print()**
- **Import** classes and functions from a library or another file, for example **import random** for random number generation
- **Comments** start with **#** and are not executed
- **Data types** can be numbers, hashmaps, lists, text
- **Classes** are your way of making your own data types
- **Operators** (+, -, %, /, //, *, **) behave differently depending on which data type
  - “A” * 10 == “AAAAAAAAAA” (Strings)
  - 5 * 10 == 50 (Integers)
  - [1] * 10 == [1, 1, 1, 1, 1, 1, 1, 1, 1, 1] (Lists)

Variables – Containers for data

Assignment



```
number = 123
```

Variable name

Data

Assignment of multiple

```
a, b = 1,2
print(f"a={a}, b={b}") >> a=1, b=2
```

Naming rules

Cannot start with a number

```
2b_or_not_2b = "that is the question"
to_be_or_not_2b = "that is the question"
```

Do not overwrite python functions

```
print = "hello"
print("will not work") (since print has been
overwritten with the str “hello”)
TypeError: 'str' object is not callable
```

Case sensitive

```
a = 123
          = Valid
A = 1234
```

Python essentials – most common data types

Text:
Strings (str)

```
x = "Hello"
type(x) == <class 'str'>
```

Booleans:
Boolean (bool) - True/False

```
x = True
type(x) = <class 'bool'>
```

Numbers:
Integer (int) - whole numbers

```
x = 123
type(x) == <class 'int'>
```

Float (float) - decimal numbers

```
x = 123.5
type(x) == <class 'float'>
```

Sequences:
List (list) - mutable

```
x = [1,"a", 3.0]
type(x) = <class 'list'>
```

Tuple (tuple) - immutable

```
x = (1,2,3)
type(x) = <class 'tuple'>
```

Text - Strings (character arrays)

Assigned with either double or single quotation marks

```
string_one = "This a string"
string_two = 'This is also a string'
```

Can be concatenated with operators

```
string_one = "Hello"
string_two = "Lexher!"
string_three = string_one + " " + string_two
print(string_three) >> Hello Lexher!
```

Comparison

```
lexher = "lexher"
print(lexher == "lexher") >> True (equals)
print(lexher == "hello") >> False (equals)
print(lexher in "lexher is great!") >> True (contains)
print(lexher not in "lexher is great!") >> False (does not contain)
```

Can be spliced (starts from 0)

```
letters = "abcdefgh"
print(letters[4]) >> e
print(letters[0:3]) >> abc
```

Useful functions and methods

```
lexher = "LeXhEr"
print(lexher.lower()) >> lexher
print(lexher.upper()) >> LEXHER
print(len(lexher)) >> 6
```

F-Strings

```
name = "Niklas Lund"
f_string = f"My name is {name}"
```

Numbers – Integers and floats

Assigned with just numbers or converted from another type with the int/float-function and reassigned with operators

```
number_one = 1.0
number_two = int("2")
```

Math with operators

```
a = 2
b = 10
print(a + b) >> 12
print(a * b) >> 20
print(b / a ) >> 5.0 (division returns float)
print(b // a) >> 5 (floor divisions returns int)
print(b % a) >> 0 (modulus)
print(b ** a) 100 (power of: 10 ^ 2)
```

CAUTION: Operations with floats

```
a = 0.1 + 0.1 + 0.1
b = 0.3
print(a == b) >> False
print(a) >> 0.30000000000000004
```

Comparison

```
a = 2
b = 10
print(b == a) >> False (equals)
print(a == 2) >> True (equals)
print(b != 2) >> True (Not equal)
print(b < a) >> False (Lower than)
print(b > a) >> True (Greater than)
```

Booleans – True / False

Assignment

```
a = True
b = False
```

All comparisons return a boolean

```
print(1 == 1) >> True
print(3 < 1) >> False
```

All datatypes have an inherent truth value

```
Integers
0 >> False
-100,100,9999 are all True

Strings
"" >> False
"sometext" >> True

Lists
[] >> False
[1,2,3] >> True
```

Flow – If, elif and else

Structure

```
suspected_age = look_at(person)
if suspected_age < 25:
    true_age = check_id()
    if true_age < 18:
        return False
    elif 18 < true_age < 40:
        return True
    else:
        print("Oh, you are older than you look!")
        return True
else:
    return True
```

Remember the colon and whitespace (indentation)

```
if name == "niklas":
print(True)

IndentationError: expected an indented block
```

Operators

```
== Equals (value)
!= Not Equals
< Less than
> Greater than
<= Less than or equal
is Equals (is the same object)
In Membership (if array contains X for example)
```

Flow – for and while loops

Structure

```
while True:
    print("this goes on forever")

i = 0
while i < 10:
    print("This goes on for 9 times")
    i += 1

for i in range(10):
    print(f"this goes on for 10 times")

letters = "abcde"
for letter in letters:
    print(letter) >> 1st loop print a, second b etc
```

Break and continue

```
while True:
    answer = input("type '123' ")
    if answer == "123":
        break
    else:
        continue
    print("I will never print")
print("We broke out of the loop!")
```

else

```
letters = "abcdefgh"
for letter in letters:
    if letter == "x":
        break
else:
    print("x not found")
```

Time to put it into action, a game!

Rules of the game:
The user has to guess a random number and the program will return if the guess is lower or higher than the number until a correct guess is given and we output the number of attempts.

Structure:

- From the **random** library import the function **randint**
- Generate a random number with the function **randint** and store it in a variable
- Create a variable that holds the number of guesses taken
- Make a **while-loop** that contains
  - Take user input with **input()** and store in a variable
  - Check if it is the correct number or if it is higher/lower
  - If correct; **break**
  - Else; output higher or lower and **continue**
- If broken out of the loop
  - Output number of guesses taken and exit

Lists

Assignment

```
people = ["Niklas", "Daniel", "Umer"]
various = [1,1.0, "text", ["other", "lists"], True]
```

Indexing and splicing [start:stop:step]

```
Indexes      0    1    2    3    4    5
             ↑    ↑    ↑    ↑    ↑    ↑
letters = ["a", "b", "c", "d", "e", "f"]
print(letters[0]) >> a
print(letters[2]) >> c
print(letters[1:]) >> ["b", "c", "d", "e", "f"]
print(letters[:3]) >> ["a", "b", "c"]
print(letters[::2]) >> ["a", "c", "e"]
```

Use **b = a.copy()** instead!

Useful functions and methods

Code:	print(numbers)
`1: numbers = [1,2]`	`1: [1,2]`
`2: numbers.append(3)`	`2: [1,2,3]`
`3: numbers.pop(0)`	`3: [2,3]`
`4: numbers.insert(1,5)`	`4: [2,5,3]`
`5: numbers.sort()`	`5: [2,3,5]`
`6: numbers.extend([1,2,3])`	`6: [2,3,5,1,2,3]`
`7: numbers.clear()`	`7: []`

CAUTION: Assignments of lists does not copy

```
a = [1,2,3]                  a = "123"
b = a                        b = a
                     !=
a.append(4)                  a += "4"
print(b) >> [1,2,3,4]        print(b) >> "123"
```

Flow – for loops and indexing

Lists

```
firstnames = ["Bill", "Elon", "Niklas", "Steve"]
lastnames = ["Gates", "Musk", "Lund", "Jobs"]
for i in range(len(firstnames)):
    print(f"i = {i}:")
    print(firstnames[i] + " " + lastnames[i])
```

```
Output:
i = 0:
Bill Gates
i = 1:
Elon Musk
i = 2:
Niklas Lund
i = 3:
Steve Jobs
```

Changing

```
numbers = [1,2,3]
for i in range(len(numbers)):
    numbers[i] *= 2
print(numbers) >> [2, 4, 6]
```

range(start, stop, step) function

```
for i in range(10,100):
    print(i) >> 10,11,12 ... 97,98,99
for i in range(1,100,2):
    print(i) >> 1,3,5 ... 95,97,99
for i in range(100,1,-1):
    print(i) >> 100,99,98 ... 4,3,2
```

Dictionaries – Key, value pairs

Assignment (can also be one-liner)

```
skills = {
    "Niklas" : ["linux", "python"],
    "Daniel" : ["linux", "kubernetes", "docker"]
}
```

Keys Values

Accessing

```
print(skills["Niklas"]) >> ["linux", "python"]
for key, value in skills.items():
    print(f"Key = {key}")
    print(f"Value = {value}")
Output:
Key = "Niklas"
Value = ['linux', 'python']
Key = "Daniel"
Value = ['linux', 'kubernetes', 'docker']
```

Cannot contain duplicate keys

```
user = {
    "firstname" : "Niklas",
    "lastname" : "Lund",
    "age" : 27,
    "mood" : "happy",
    "mood" : "angry"
}
print(user["mood"]) >> angry
```

Functions – Declaring and calling

Declaration

```
def function_name(argument_one, argument_two,keyword_argument=True):
    global global_variable
    global_variable = 2
    local_variable = 3
    return global_variable + local_variable
```

Calling

No arguments

```
def hello():
    print("Hello!")
hello() >> "Hello!"
hello("Niklas") >> ERROR
```

One argument

```
def hello(name):
    print(f"Hello {name}!")
hello() >> ERROR
hello("Niklas") >> "Hello Niklas!"
```

One argument with a default value

```
def hello(name="User"):
    print(f"Hello {name}!")
hello() >> "Hello User!"
hello("Niklas") >> "Hello Niklas!"
```

Functions – Returning and scope

Scope

```
x = 5
def set_x(number):
    x = number
set_x(50)
print(x) >> 5
```

```
x = 5
def set_x(number):
    global x
    x = number
set_x(50)
print(x) >> 50
```

Returning values

```
def returns_two():
    return 2
print(1 + returns_two()) >> 3
```

```
def both_names(name):
    firstname, lastname = name.split(" ")
    return firstname, lastname
print(both_names(“Niklas Lund")) >> "Niklas”, “Lund"
```

Classes

Declaration

Special Init method

```
class Person():
    def __init__(self, name, skills):
        self.name = name.capitalize()
        self.skills = skills

    def describe(self):
        print(f"Hi, my name is {self.name}")
        print("My skills are:")
        for skill in self.skills:
            print(f"\t{skill}")

    def learn(self, new_skill):
        print(f"{self.name} learned {new_skill}!")
        self.skills.append(new_skill)
```

Class Methods

Using

```
niklas = Person("Niklas", ["Linux", "Python"])
daniel = Person("Daniel", ["Linux", "Kubernetes", "Docker"])
daniel.describe()
niklas.learn("Openshift")
niklas.describe()
```

Output

```
Hi, my name is Daniel
My skills are:
        Linux
        Kubernetes
        Docker
Niklas learned Openshift!
Hi, my name is Niklas
My skills are:
        Linux
        Python
        Openshift
```

Example, using classes to make a deck of cards

```python
class Card():
    def __init__(self, value, suit):
        self.value = value
        self.suit = suit

    def __repr__(self):
        translate = {
            11 : "jack",
            12 : "queen",
            13 : "king",
            1 : "ace"
        }
        if self.value > 10 or self.value == 1:
            return f"{translate[self.value]} of {self.suit}"
        else:
            return f"{self.value} of {self.suit}"
```

```python
class Deck():
    def __init__(self):
        suits = ["hearts", "clubs", "spades", "diamonds"]
        values = [1,2,3,4,5,6,7,8,9,10,11,12,13]
        #Initiating a full deck
        self.cards = []
        for value in values:
            for suit in suits:
                self.cards.append(Card(value,suit))

    def get_random_card(self):
        first_card = self.cards.pop(0)
        self.cards.append(first_card)
        return first_card

    def shuffle(self):
        random.shuffle(self.cards)
```

Popular Libraries

Web Development

- FastApi
- Flask
- Django

Data analyzing

- NumPy
- Pandas

From the Standard Library

- requests - Send HTTP requests
- math - Math based operations
- random - Random number generation and other
- itertools - Iterator functions and helper
- threading - Multithreading processes
- os - Operating system interface
- sys - Access to interpreter
- datetime - Dates and time
- time - Timing functions and sleep

Thank you!