Introduction to programming in Python

Jules Kreuer

Uni Tübingen

fsi@fsi.uni-tuebingen.de
contact@juleskreuer.eu

March 18, 2022

Based on:

Ana Bell, Eric Grimson, and John Guttag.
6.0001 Introduction to Computer Science and Programming in Python.
Fall 2016. Massachusetts Institute of Technology: MIT OpenCourseWare
https://ocw.mit.edu.

Nick Parlante, John Cox, Steve Glassman, Piotr Kaminksi, Antoine Picard.
Google's Python Class.
July 2015. Google LLC

Part 1: Hello World

- Introduction
- Installation
- REPL

Break

Part 1: Hello World

- Introduction
- Installation
- REPL

Break

Part 2: Basics

- Common operators
- Data types, type-casting
- Lists, dicts
- Control flow: for, while, break, continue

Break

Part 3: Abstraction

- Functions, Imports, variable scope
- lambda
- Files / IO
- Objects, Classes
- Exceptions

End

Resources

https://juleskreuer.eu/projekte/python/

Part 1: Hello World

```
jules@T480:~$ python3
Python 3.8.10 (default, Nov 26 2021, 20:14:08)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license"...
>>> a = 5
>>> a
5
>>> a = "Hello World"
>>> a
'Hello World'
>>> a + "!"
'Hello World!'
>>>
```

Installation / REPL

`https://www.python.org/downloads/`
Debian / Ubuntu: *sudo apt install python3*

Type in your shell: *python3*

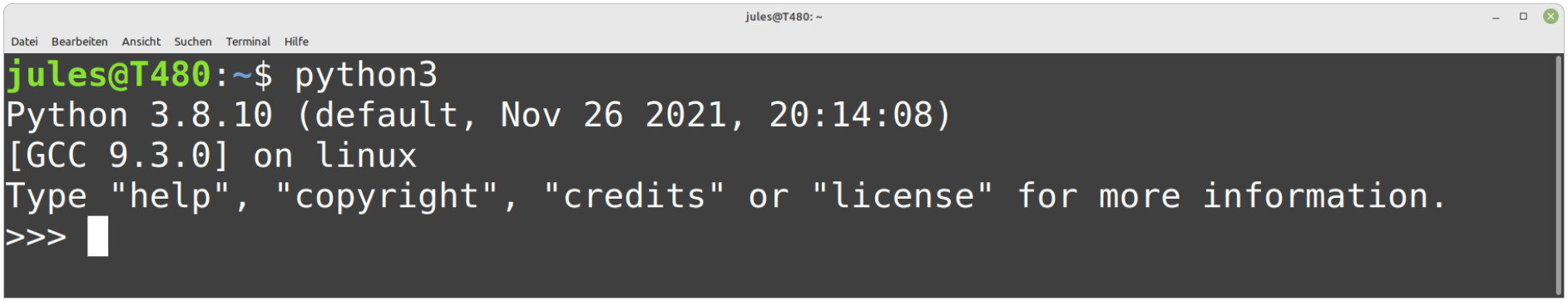

Figure: Python3 REPL

Running code

- REPL
- python3 file args

Example

```
python3 hello-world.py
```

Combining Editor and Interpreter

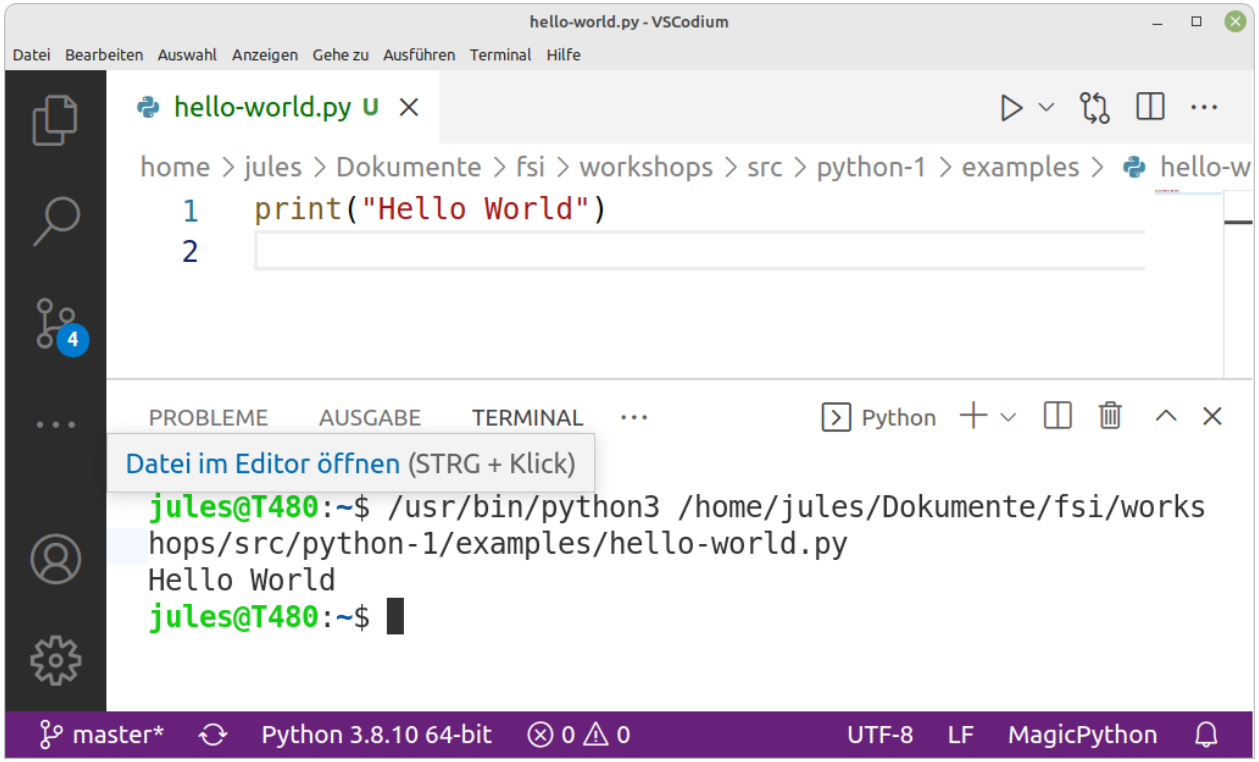

Figure: VS Codium

Possible IDEs / Editors:

- VS Codium: `https://vscodium.com/`
- PyCharm: `https://www.jetbrains.com/pycharm/`
- Atom: `https://atom.io/`
- ...

hello-world.py

- Content: `print("Hello World")`
- Run it!

Basic operators and types

Just like 'any other' language.

Math

```
s = (a + b - c) / d * e
p = a ** 2 # a to the power of 2
b = a^2    # bitwise shifting
m = a%2    # mod
```

Numeric types

```
int, float, complex
i = 1 = int("1") = int(1.0)
f = 4.2
c = 4+2j
```

Strings

```
s = "Hello " + "World"
c = "A" * 10 + "HHHH"
S = s.upper()
length = len(S)    # Returns Integer
pos = s.find("W") # Return Integer (Position of first W)
```

Text types

```
str
s = str(1)
```

Booleans

```
a = (True or False) and not False
```

Boolean types

```
bool
t = bool(1) = bool("Not Empty")
f = bool(0) = bool("")
```

Comparision

`<, >, ==, !=, <=, >=`

Example

```
t = 3 < 5
t = not "A" == "B"
f = 4.2 == 2
f = 0 == "Hello" # Comparision in between types is possible
```

01-types.py

Exercise

Desired output: 'The sum of 41.8 and 0.2 is 42'.

Use following variables:

```
i = 41.8
f = 0.2
prefix = "The sum of "
```

Lists

Mutable, dynamic in length, non-homogenous, ordered

```
aList = [1, 2, 3, 4, "What?", 6]
aList[0]                  # -> 1
aList[4:]                 # -> ['What?', 6]
aList[1::2]               # -> [2, 4, 6]
aList[-1]                 # -> 6
aList.append(7)           # -> [..., 6, 7]
aList.extend([8,9])       # -> [..., 6, 7, 8, 9]
aList[0] = "New Zero"
general form: [from:to:step/order]
```

Lists

Mutable, dynamic in length, non-homogenous, ordered

```
aList = [1, 2, 3, 4, "What?", 6]
aList[0]                  # -> 1
aList[4:]                 # -> ['What?', 6]
aList[1::2]               # -> [2, 4, 6]
aList[-1]                 # -> 6
aList.append(7)           # -> [..., 6, 7]
aList.extend([8,9])       # -> [..., 6, 7, 8, 9]
aList[0] = "New Zero"
general form: [from:to:step/order]
```

Tuples

Non-Mutable, fixed length, non-homogenous

```
aTuple = ("A", "a", 1)
a[0]              # -> "A"
```

Dicts

Mutable, dynamic in size, non-homogenous, unordered[a]

```
d = {"key": "value", 1: 3}
d["key"]      # -> "value"
d["new"] = 2 # Insert new value to d
d.keys()      # -> ["key", 1, "new"]
d.values()    # -> ["value, 3, 2]
d.items()     # -> [("key", "value"), (1, 3), ("new", 2)]
```

[a] Somehow..

See: https://docs.python.org/3/tutorial/datastructures.html

Control flow: if, for, while, break, continue

Regular control flow with if:

```
if condition:
  doThis()
elif cond2:
  doThat()
else:
  otherWise()
```

Looping has two different approaches:

while / condition

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

Looping has two different approaches:

while / condition

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

for / iterable

```
for element in iterable:
  print(e)
```

Iterables: something with an order and members.

Example

```
tuples = (0, 1,2,3,4)
lists  = [0, 1,2,3,4]
string = "Hello World"
dicts = {"a", 2}
range(0, 6, 2) # start, stop, interval
               # somehow comparable to [0,2,4]
file objects
...
```

for / iterable

```
for element in iterable:
  print(e)
```

for / iterable

```
for element in iterable:
  print(e)
```

Example

```
for i in range(5):
  print(e) # 0, 1, 2, 3, 4
for c in "Hello World":
  print(e) # Every char
for k in {"k": "v", "k2": "v2"}:
  print(k) # Only the keys
for k, v in {"k": "v", "k2": "v2"}.items():
  print(k, v) # Unpacking
```

Unpacking:
- Object with ordered members
- Number of vars equal to members[1].

Example

```
a, b, c = (1, 2, 3)
a, b,   = [1,2]
```

[1] `x, *xs = [1, 2, 3, 4]` → `xs = [2,3,4]`

Exit the loop early?

Break

```
while True:    # works for "for i in .." aswell
  doThis()
  if exitCondition:
     break
```

Skip to the next element?

Continue

```
for i in range(4):
  if i == 2:
    continue
  print(i)
-> 0, 1, 3
```

02-number-guess.py

Exercise

Implement a basic python number guessing game.

1. Generate a random number.
2. Ask for a guess.
3. Check if guess was correct.
4. If not, say if number was smaller / larger
5. Repeat from step 2, but only 8 times max.

Use following functions:

```
from random import randint
randint(0,1024)  # random integer N such that a <= N <= b
input("Number?") # Takes input from user
```

03-lists.py

Exercise

Understand how to uses loops and lists.

E1:

Print the last element of list l1:

```
l1 = ["first", "middle", "last"]
```

E2.1:

Print every second element of list l2_1

Without loop.

```
l2_1 = [0,1,2,3,4,5,6,7,8,9]
```

....

by value / by reference

Example

```
l1 = [1,2,3,4]
l2 = l1
l1.append(5)
print(l2)
[1, 2, 3, 4, 5]
```

See post from 'Russia Must Remove Putin':
https://stackoverflow.com/a/46939443/5410925

Part 3: Abstraction

Functions:

- Decomposition of Code into parts
- Function acts like a black bock

```
def is_even(i):
    """
    Computes if an integer is even.
    Input:
            i: int
    Returns:
            even: bool, result.
    """
    even = (1 % 2) == 0
    return even
```

Part 3: Abstraction

Functions:

- Decomposition of Code into parts.
- Function acts like a black bock.

```
def is_even(i):                  <- keyword, name(parameters)
   """                           _
   Is integer is even?            |
   Input:                         |
     i: int                       |> DocString
   Returns:                       |
     even: bool, result.          |
   """                           _|
   even = (1 % 2) == 0            <- Computation
   return even                    <- Return (Optional)
```

Example

```
def noReturn(a, b):
  print(a)

def optionalArgument(a, b=0):
  return

def optionalReturn(x):
  if x < 5:
    return True

def polymorphicReturn(x):
  if x < 5:
    return True
  return x
```

Modules / Import:

- Full: `import moduleName`
- Partial: `from moduleName import subModule`
- File in same directory: `import filename`
- A lot of standard libraries:
  - Math: random, statistics, math
  - Time: time, datetime
  - OS/IO: argparse, os, pathlib, sys
  - Network: urllib3
- See: `https://docs.python.org/3/library/index.html`
- Extended standard: numpy, pandas, ...

Scope:

- Which variables are visible from which part of the code.
- From outer to inner.

Example

```
def useX(y):
  return y + x

def modifyX(y):
  x = y + x

x = 10
y = useX(5)
modifyX(y)
print(x)
```

Scope:

- Which variables are visible from which part of the code.
- From outer to inner.

Example

```
def useX(y):
  return y + x

def modifyX(y):
  x = y + x <- Assignment forces x to be local variable
            -> Error: local variable 'x'
               referenced before assignment
x = 10
y = useX(5)
modifyX(y)
print(x)
```

scope.py

Example

```
def g(x):
  def h():
        x = 'abc'
  x = x + 1
  print(f"x in g: {x}")
  h()
  return x

x = 3
print(f"x at position 2: {x}")
z = g(x)
print(f"z at position 1: {z}")
```

lambda functions:

- good for single use functions
- usually defined inline
- useful for currying

Example

```
l = [0,1,2,3,4,5,6,7,8,9]
lPow = map(lambda x: pow(x,2), l)

equiv to: [pow(x,2) for x in l]
```

File reading:
- Files need to be opened and closed.
- We don't want to handle that...

Example

```
with open("path/file.txt", "r") as f:
  line = f.readline()
  content = f.read() # reads everything

with open("path/file.txt", "r") as f:
  for line in f:
    print(line)
```

Modes:
- r, read

File writing:

Example

```
with open("path/file.txt", "w") as f:
  f.write("string")
  f.write("\n")
  f.writelines(["line1", "l2", "l3"])
```

Modes:

- w, overwrite / write
- a, append
- x, write, error if file exist

06-wordle.py

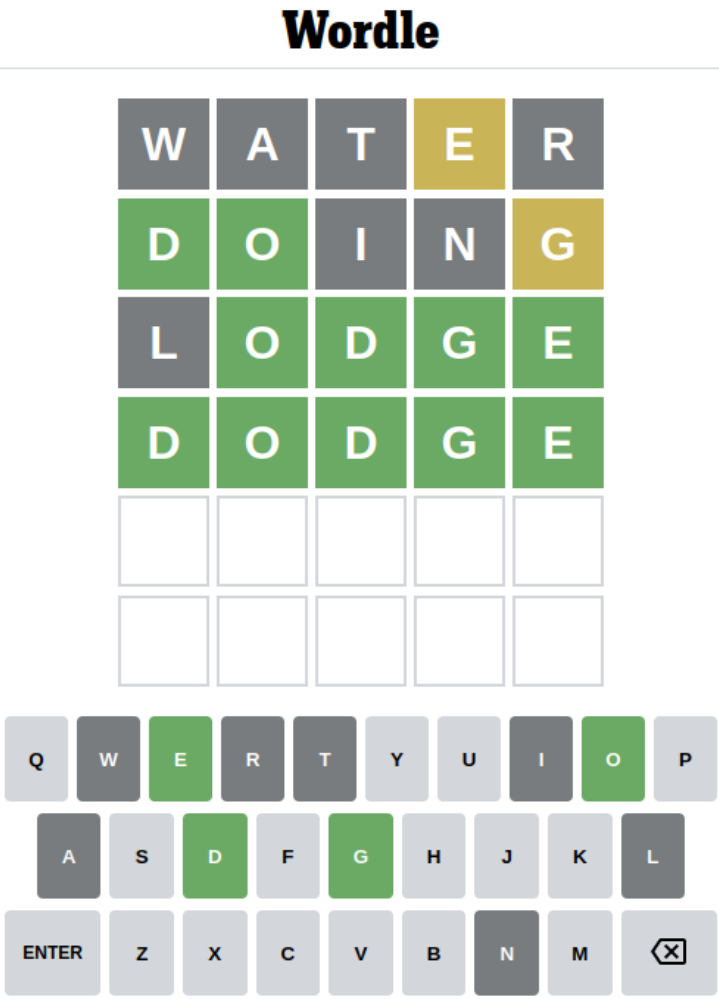

Figure: Wordle by NYTimes, `https://www.nytimes.com/games/wordle`

06-wordle.py

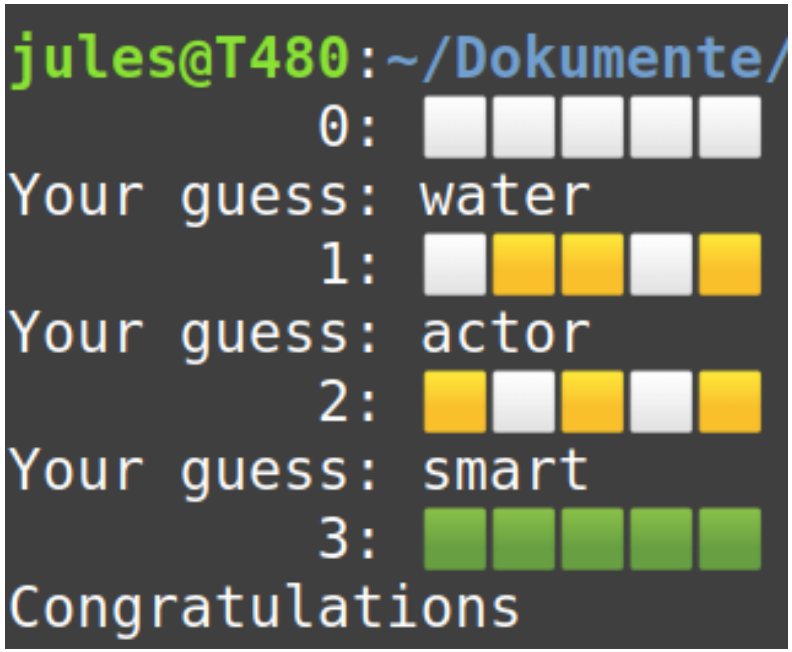

Figure: Our goal.

Objects / Class

- Object is instance of an Class.
- Has properties and methods.
- Everything is a object.

Class

```
class MinimalClass():
  def __init__(self):
    pass
```

Object

```
x = MinimalClass()
```

Functions defined in a class:
- are applied on an object.
- requires at least one argument (self).

Example

```
class Counter():
  def __init__(self, x):
    self.x = x

  def addOne(self):
    self.x = self.x + 1

c = Counter(0)
c.addOne()
print(c.x)
```

Functions defined in a class:
- are applied on an object.
- requires at least one argument (self).

Example

```
class Counter():
  def __init__(self, x):
    self.x = x          <- property

  def addOne(self):     <- self required
        self.x = self.x + 1

c = Counter(0)          <- calling __init__
c.addOne()              <- alling addOne()
print(c.x)              <- accessing property
```

Style / Information hiding:
- Use getter / setter method outside of class
- Information hiding is NOT possible

Inheritance:

Example

```
class Animal():
  def __init__(self, name):
    self.name = name

class Cat(Animal):
  def speak(self):
        print("Meow")

Cat("Kleopatra").speak()
```

Magic Methods[2]:

- Adds 'magic' to a class.
- Start / end with __ . Example: `__init__`
- Comparison, Type Conversion, Representation, Context

Example

```
Cat("Kleopatra") == Cat("Kleopatra")
--> False (two different objects)
```

[2] A complete guide: `https://rszalski.github.io/magicmethods/`

Magic Methods[3]:

- Adds 'magic' to a class.
- Start / end with __ . Example: `__init__`
- Comparison, Type Conversion, Representation, Context

Example

```
Cat("Kleopatra") == Cat("Kleopatra")
--> False (two different objects)

class Animal():
  def __eq__(self, other):
          return self.name == other.name

--> Cat("Kleopatra") == Cat("Kleopatra") -> True
```

[3] A complete guide: https://rszalski.github.io/magicmethods/

Comparision

Equality: `__eq__(self, other)`
Greater than: `__gt__(self, other)`
Less than: `__lt__(self, other)`
...

Arithmetic

Addition: `__add__(self, other)`
Multiplication: `__mul__(self, other)`
...

Sequences

Iterator: `__iter__(self)`
Reversed: `__reversed__(self)`
...

07-objects.py

Exercise

Goal: Implement a Vector-Class with following properties:

1: Holds values for x, y, z
   Example: V1 = Vector(1,2,3)
2: Equal `__eq__`
3: Print `__str__`
4: Vectors can be added, this will return a new vector
   Example: `V1 + V1 -> Vector(2,4,6)`
5: Extend code so that `V1.add(Vector(2,3,4))` will mutate V1.
   We do not want to see duplicated code.
   Remember: use deepcopy to clone an object.

Exceptions

- Error will raise an exception.
$\rightarrow$ terminates programme.
- We can catch and raise them.

Raise

```
raise Exception("Your error message")
```

Catch

```
try:
  y = 1 / x
except:
  y = float("-inf")
else:     <- optional
  print("everything ok")
```

Exception types[4]:

Be more specific while raising / catching exceptions!

Types

- ZeroDivisionError
- IndexError (lists)
- KeyError (dicts)
- TypeError (wrong type, forgot to cast?)

Example

```
try:
  y = 1 / x
except ZeroDivisionError:
  y = float("-inf")
except Exception as e:
  print("Other error:" , e)
```

[4] Complete list: `https://docs.python.org/3/library/exceptions.html`

Thank You!