EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Faculty of Science
Department of Computer Science

Vorkurs Programming - Informatik 4 Lifescientists

Introduction into Programming & Scripting in Python

Part 1

RULES

1) We will go through the course alternating lectures and coding sessions !
2) Feel free to interrupt and ask questions any time !
3) There are no stupid questions or things you should know !

…

CHAPTERS

1) Basic terms
2) First Steps in Python
3) Data Types and more e.g.:
   - variables
   - data types
   - functions
4) Conditional Programming and loops e.g.:
   - if statements
   - for loops

…

Basic terms

What is an algorithm?

What is a computer program?

What is a programming language and source code?

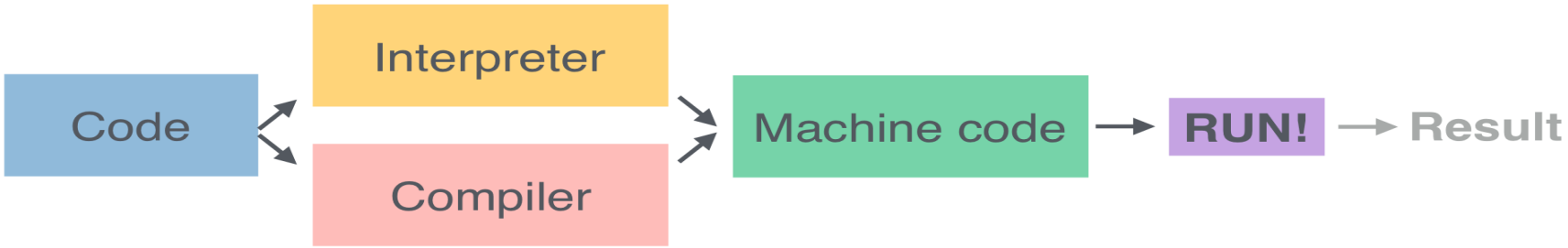
How is the code machine readable?

Code → Machine code → **RUN!**

```
def read_details(fname):
    """ read details """
    print('reading', fname)
    f_handle = open(fname, 'r')
    text = f_handle.read()
    f_handle.close()
    out_dct = {}
    for reg in text.split()[1:]:
        scaff, hits = parse_reg(reg)
        out_dct[scaff] = hits
    return out_dct
```

```
01000100101010
11010100110010
01010100101011
01000100101010
11010100110010
01010100101011
```

How is the code machine readable?



What is an integrated development environment (IDE)?

What is an integrated development environment (IDE)?

First Steps in Python

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.

Do I have to like "Monty Python's Flying Circus"?

No, but it helps. :)

mamba
ANACONDA
snakemake

Python is interpreted

Hence, no need for compilation

Run the script

```
cloudybay:~ SPatz$ python3 python_script.py
```

→ remember LINUX commands

We use the Python-IDE PyCharm, so no need to run a script via command line.

Requirements

Please install:

- **Python 3.7 or 3.8** (Download)
- **PyCharm** (Community Edition) - a Python IDE

You may apply for a free professional edition of the JetBrains tools (PyCharm, IntelliJ ..) with the student email address (JetBrains Students).

Environments

What are environments?

→ Containerized Python instances

Why use environments?

→ Several Projects – Different Packages

What kind of environment systems are there for Python?

→ virtualEnv, conda environments (what we will be using)

Environments

How are they created?

→ Via Pycharm

→ Via console:

```
conda create --name envName python=3.X
```

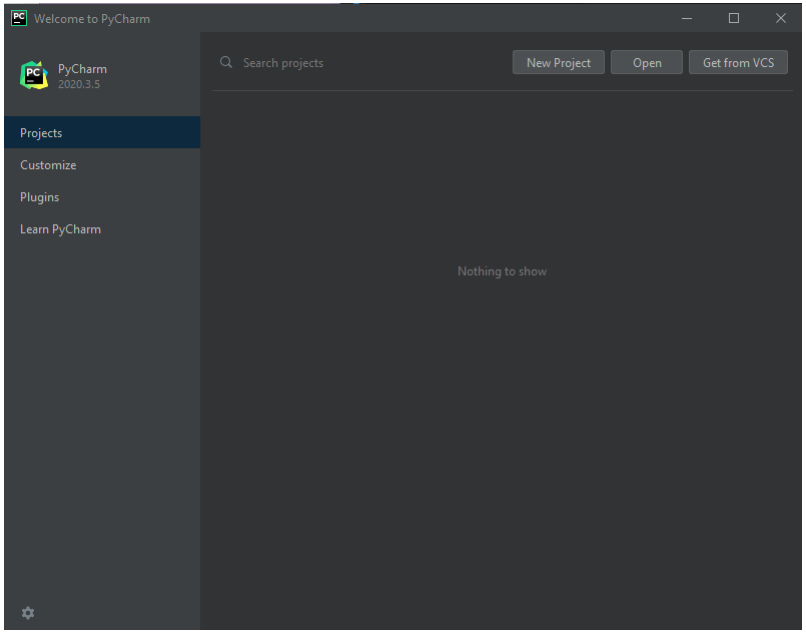
→ From .yml file:

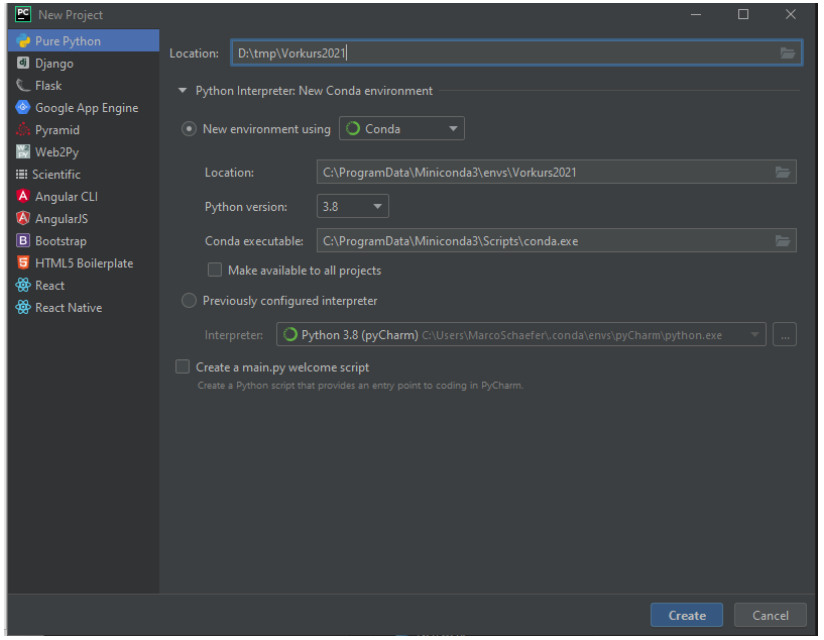
```
conda env create -f environment.yml
```

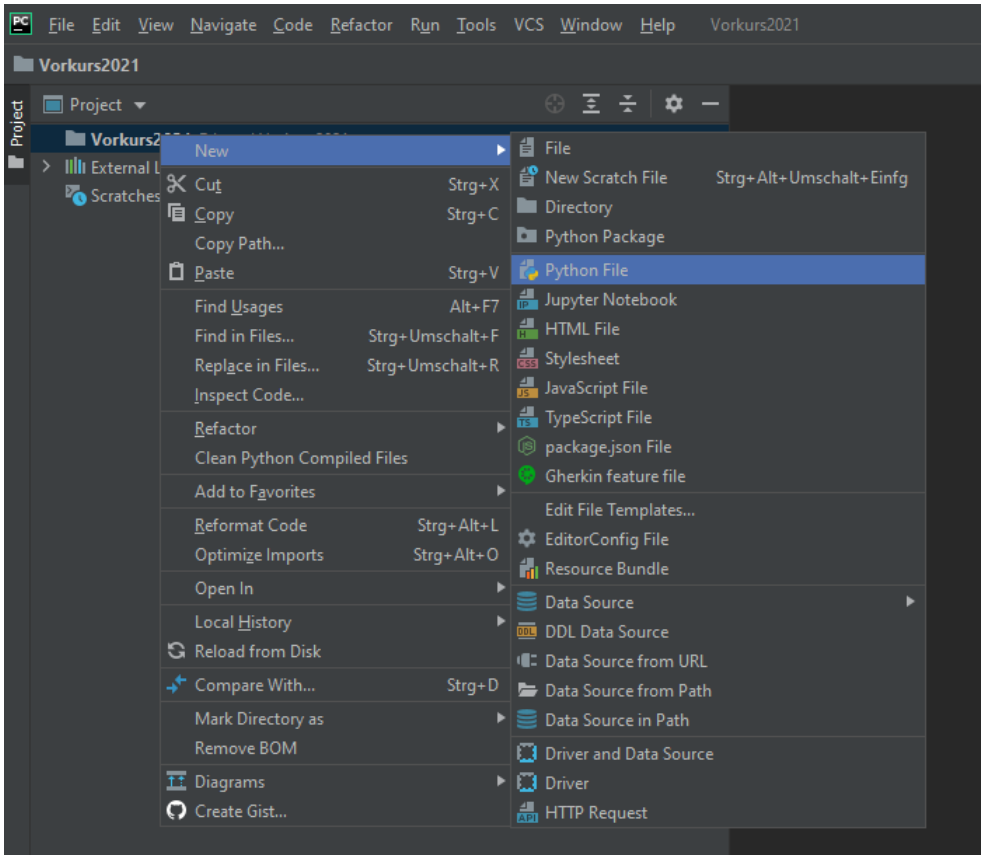
How to export?

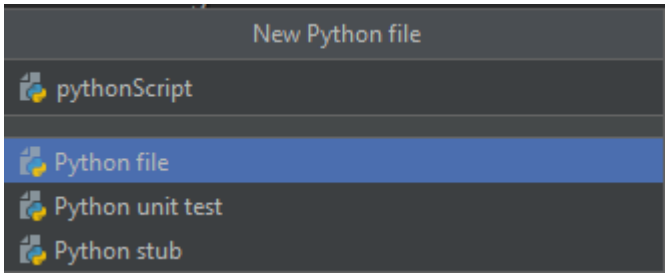
→ From the conda env:

```
conda env export > environment.yml
```

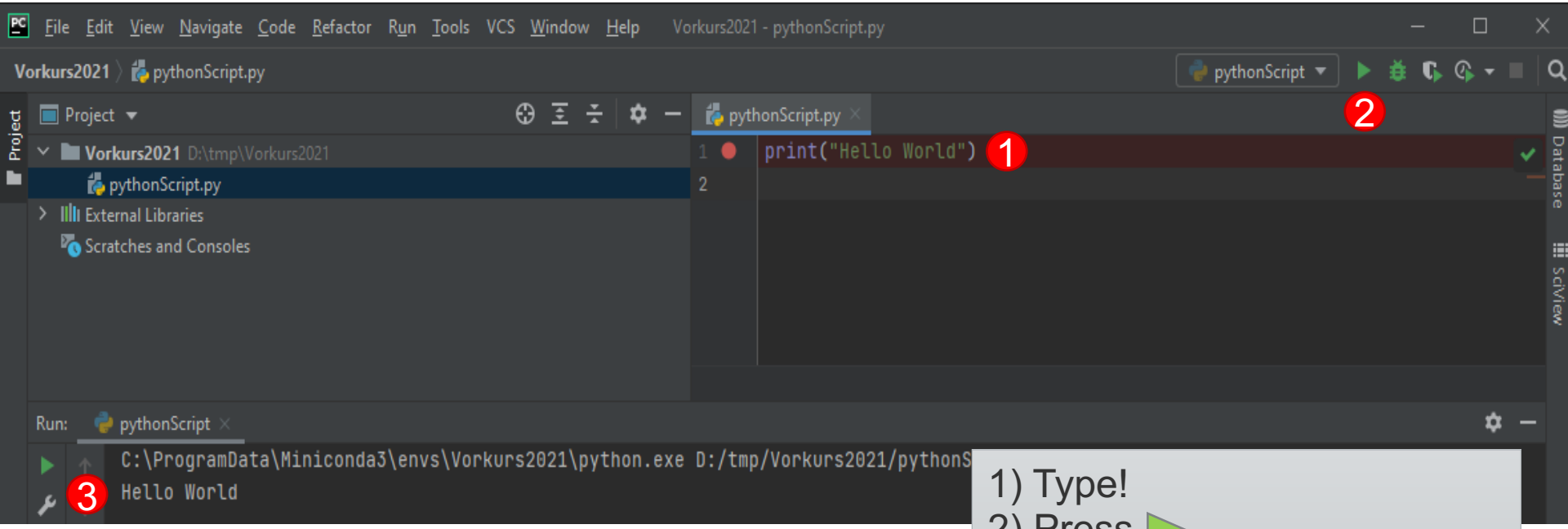








Hello World in the python interactive shell …



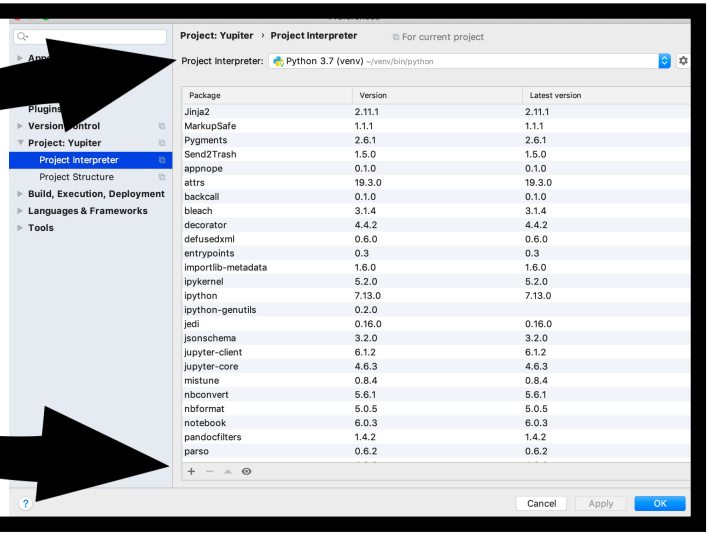
1) Type!
2) Press ▶
3) Interactive shell will open.

You may check “Terminal”, that is like the LINUX Bash/Shell-Terminal

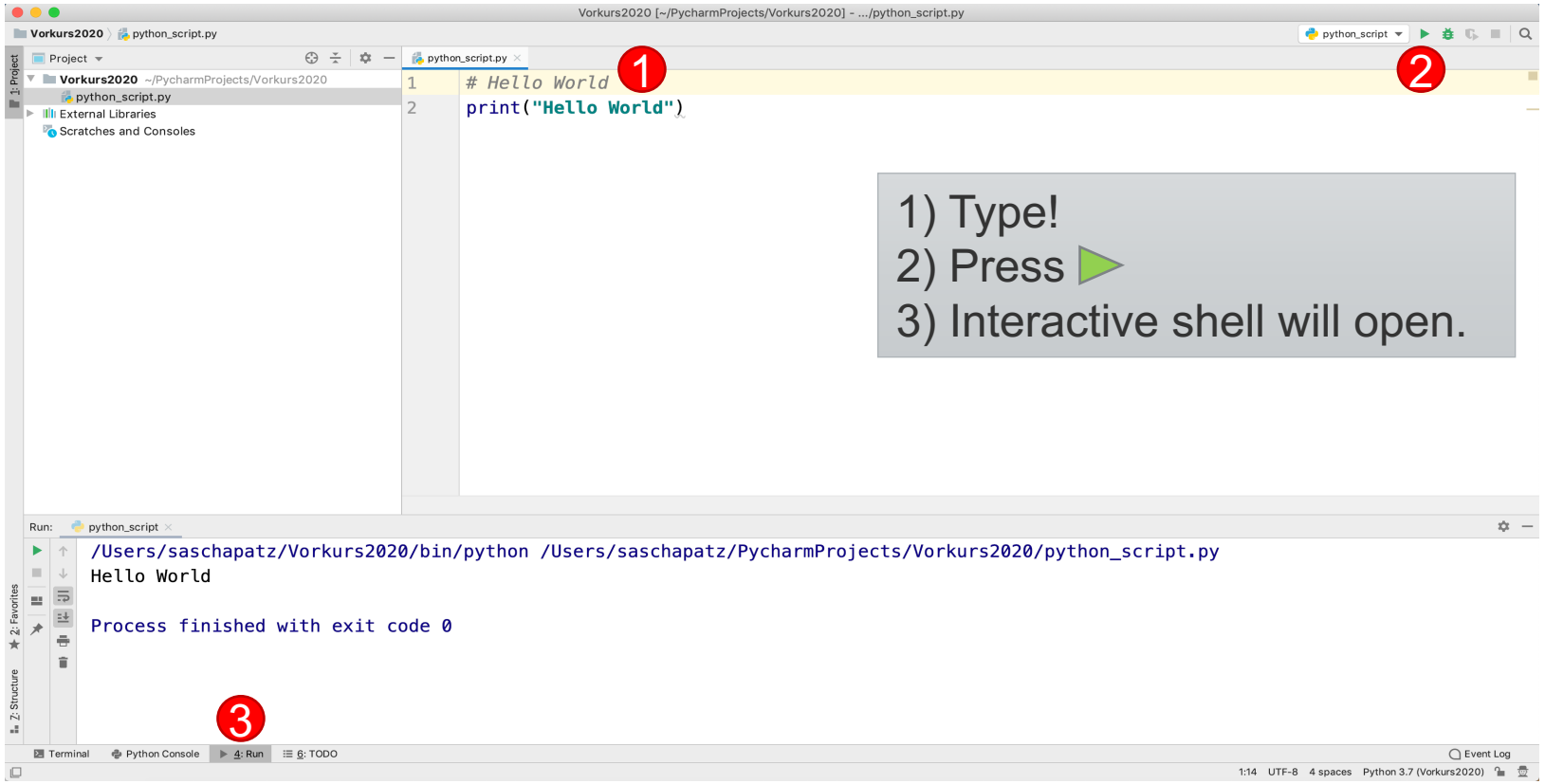
Any issues, running the Python Script?

I some cases you have to **specify the Python Interpreter in PyCharm explicitly** for running scripts. To do so, check Settings, Preferences or Configurations for your Project.

There you can add / install new packages like bio or biopython as well.



How to use Comments?



Comments can be written behind active code: `print("Hello World")  # Hello World`

Test yourself?

python_script.py

```
1  # Hello World
2  print("Hello World")
3  print('Welcome to Sequence Bioinformatics')
4
```

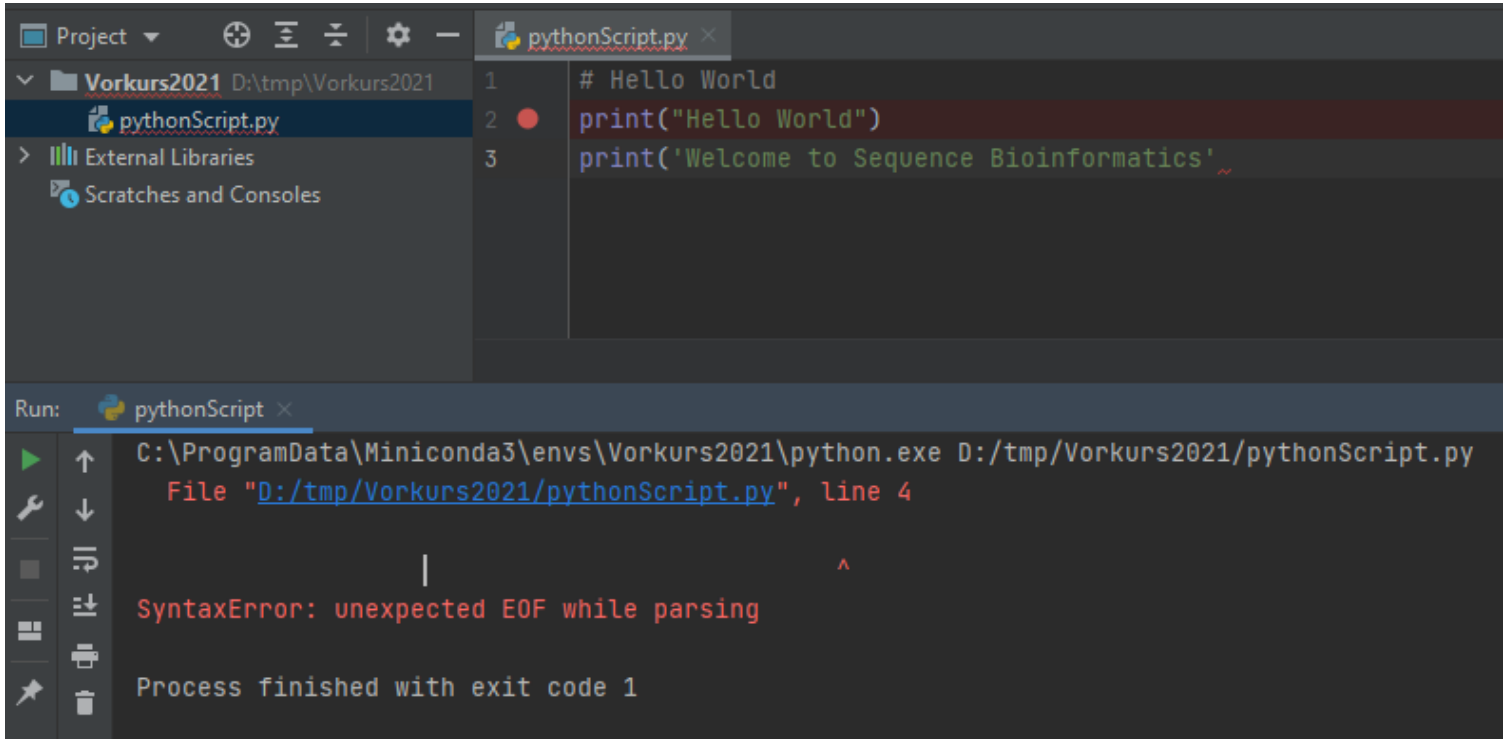
Take care of the parentheses!

What happens, if you delete the last bracket of one line?

Get used to error messages and try to understand them!

You should see that message:

Error Messages ...



It will take time to understand the meaning of each error, but try to look up any arising error!

Here, the SyntaxError tells you, that there is a typo somewhere in your code. It tries to show you the line (4) and position (^).

Time to try and error Q and A!

Data Types and More

What is a Variable?

How to declare Variables?

python_script.py

```
1  # Hello World
2  print("Hello World")
3  print('Welcome to Sequence Bioinformatics')
4
5  a=4
```

Type the new line and press ▶ for running new code!

Why is there no output?

How to print() each variable into the terminal?

Declare more Variables ...

python_script.py

```
1  # Hello World
2  print("Hello World")
3  print('Welcome to Sequence Bioinformatics')
4
5  a=4
6  b = 3.0
7  c = 'what kind of typing is that?'
8  d = "appostrophes are not important! Can be \' or \""
9  e = True
10 f = None
```

Type line by line and Press ▶ to run the code!

Add print commands for each variable to get a feeling how the code is interpreted!

Use comments to inactivate code and run (▶) again!

What are the different data types?

python_script.py

```
1  # Hello World
2  print("Hello World")
3  print('Welcome to Sequence Bioinformatics')
4
5  a=4
6  b = 3.0
7  c = 'what kind of typing is that?'
8  d = "appostrophes are not important! Can be \' or \""
9  e = True
10 f = None
```

Check the types of each variable using: `type(a)` !
Write the returning information into the terminal by applying print().

Use comments to add the data type behind active code!

Look up more data types in Python, and report later in the Q&A Session!

What is

type casting?

You are able to convert ("cast") types into each other …

python_script.py

```
1  # Hello World
2  print("Hello World")
3  print('Welcome to Sequence Bioinformatics')
4
5  a=4
6  b = 3.0
7  c = 'what kind of typing is that?'
8  d = "appostrophes are not important! Can be \' or \""
9  e = True
10 f = None
11 g = str(a)
12 h = int(g)
```

Type and run!

Check the types of variable ***g*** and ***h*** by printing them into the terminal!

Add the types as comments, as done before!

Look up more casting options!

Provoking an Error Message ...

```
15  g = str(b)
16  print(type(g))
```

```
h = int(g)
```

Type both lines and run ( )!

What happened with the variable g, that was assigned before by a (g = str(a))?

Provoking an Error Message ...

```
15  g = str(b)
16  print(type(g))
```

```
h = int(g)
```

Type both lines and run (▶)!

What happened with the variable g, that was assigned before by a (g = str(a))?

You should see that message:

```
Traceback (most recent call last):
  File "/Users/saschapatz/PycharmProjects/Vorkurs2020/python_script.py", line 13, in <module>
    h = int(g)
ValueError: invalid literal for int() with base 10: '3.0'
Hello World
Welcome to Sequence Bioinformatics

Process finished with exit code 1
```

Provoking an Error Message ...

```
15    g = str(b)
16    print(type(g))
```

```
h = int(g)
```

Type both lines and run (>)!

What happened with the variable g, that was assigned before by a (g = str(a))?

You should see that message:

```
Traceback (most recent call last):
  File "/Users/saschapatz/PycharmProjects/Vorkurs2020/python_script.py", line 13, in <module>
    h = int(g)
ValueError: invalid literal for int() with base 10: '3.0'
Hello World
Welcome to Sequence Bioinformatics

Process finished with exit code 1
```

Try to understand the error message, and why b cannot be convertet to an int!

Again, understand how the code is interpreted, as some lines of code were running!

How to inactivate the last two lines, that causes an error?

How to inactivate the last two lines, that causes an error?

You think about using # ? → Good!

How to inactivate the last two lines, that causes an error?

You think about using # ? → Good!

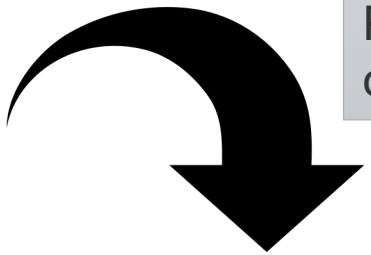
You think about using ” ... ”? → Great!

Yeah, there is another possibility to add block comments (' ' ' . . . ' ' '), see next slide.

How to create a block comment?

```
15  '''
16  g = str(b)
17  print(type(g))
18  '''
```

Block comments can be un- / collapsed in PyCharm, by clicking on - or +.



```
15  ''' ... '''
19
```

Time to try and error Q and A!

Operators and Lists

Simple Operations ...

```
# Simple Operations
a=4
b = 3.0
g = str(a)
h = int(g)

print(a+a)
print(b+b)
print(a+b)
print(g+g)
print(h+h)
```

Addition

String concatenation

Run the code and check each output!

Take care of concatenation!

```
1   x = 'banana'
2   print((x+x))
3
4   bananabanana
```

More simple Operations ...

`print(a-a)`	Substraction
`print(a*a)`	Multiplication
`print(a/a)`	Division
`print(a**a)`	Power

More simple Operations …

```
print(a-a)
print(a*a)
print(a/a)
print(a**a)
```

Multiplication

Works also with strings

```
1   x = 'banana'
2   print((x*3))
3
4   bananabananabanana
```

More simple Operations …

```
print(a-a)
print(a*a)
print(a/a)
print(a**a)

print(a%a)
print((a+1)%a)
print((a+2)%a)
print((a+3)%a)
print((a+4)%a)
print((a+a)%a)
print((2*a)%a)
```

Do you know the Modulo Operator?

Basic logical expressions and operators ...

```
# Logical Expressions and Operators
print(a==a)
print(a==b)
print(a != b)
print(a<b)
print(True)
print(False)
print(True and False or True)
print(True and (False or True))
print(True and (False or not True))
```

Equal

Not Equal

Greater

AND / OR / NOT

Using Lists ...

```
# Declare an empty list
a_list = []
print(a_list)

# Lists can contain any types of variables
my_list = [a, b, c, d, e, f, g, h]
print(my_list)

# You can always check the type of the variable
print(type(my_list))
```

Lists and its indices ...

`animal_lst =` [ ,  ,  , ]

indices: 0 1 2 3

```
# Get items of a list by index
print(my_list[0])
print(my_list[1])
print(my_list[2])
print(my_list[3])
print(my_list[0:3]) # slicing [inclusive:exclusive]
print(my_list[1:3])
print(my_list[2:])
print(my_list[:3])
print(my_list[:-1])
print(my_list[::-1]) #reverse order

# Get the length of a list
print(len(my_list))
```

Understand each line!

Look up “slicing” of lists!

Check the last index of the list!
Why the length of a list differs from the last index of the list?

Lists and its indices ...

```
# Get the index of an item
print(my_list.index(3.0))
```

Understand each line!

What happens, if an item occurres twice?

How to add an item to a list?
Add the string “hello” to my_list.
Check its index, call it by index and report on the length of the my_list again.

Using Dictionaries ...

```
# Dictionaries
# Declare an empty dictionary
a_dictionary = {}
print(a_dictionary)

# Dictionaries contain keys and assigned values, both of various types
my_dictionary = {'one key': 'one value', 2: '4', 12: 'vv', "number":10}
print(my_dictionary)

# Adding a new key and its value (here a list) to the dictionary
my_dictionary['unique key'] = ['any value']
print(my_dictionary)
```

Type line by line and Press ▶ to run new code!

Using Dictionaries …

```
# Access all keys or values of a dictionary
print('Keys', list(my_dictionary.keys()))
print('Values', list(my_dictionary.values()))

# Get a value of a key
print(my_dictionary['unique key'])
```

Type line by line and Press ▶ to run the code!

Check the type of ”my_dictionary”!

How does the dictionary differs from a list regarding sorting?

Time to
Try and Error
Q and A!

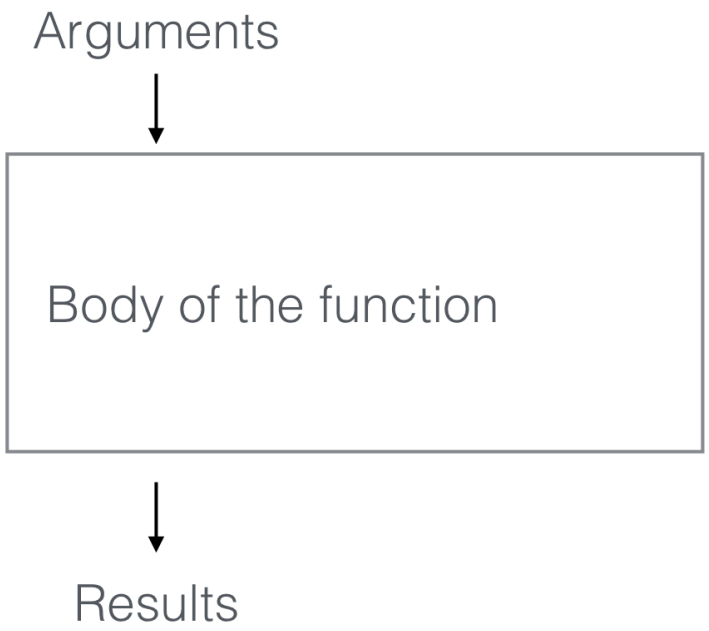
Functions

What is a Function?

Did you notice, that you have applied different functions until now?

print()
len()
index()
...

Structure of a function ...



Declare a function ...

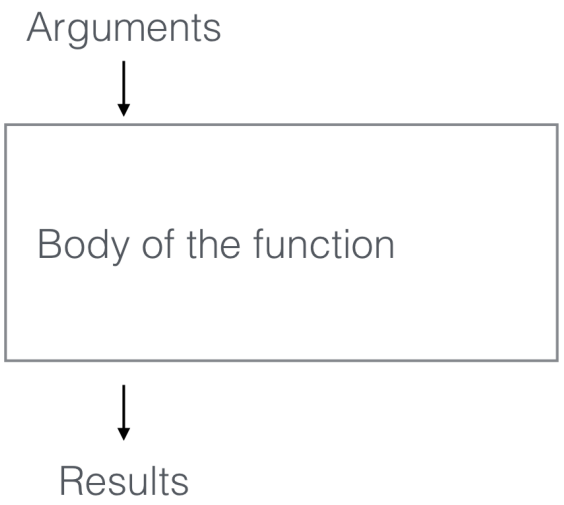
Keyword **declaring** a function

Name of a function

Arguments given to a function

Do not forgot about the **colon**

```
def print_me(internal_variable):
    print(internal_variable)
```



BODY OF A FUNCTION

Forms a block, that can be distinguished from the header through an **indent** (e.g. a tab space).

Contains any **operation**.
Uses the **input argument** (internal variable).

May define the **return value** of function (see later).

Write your own Function ...

```
# Functions
# Declare a simple print function
def print_me(internal_variable):
    print(internal_variable)
```

Type and run the code!

What happens?

Execute (call) your own function …

```
# Declare a simple print function
def print_me(internal_variable):
    print(internal_variable)

# Call a function
print_me("Hello!")
```

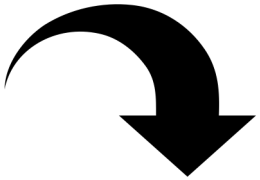
Type and run the code again!

The string “Hello!” is handed over /assigend to the “internal_variable” similar to internal_variable = “Hello!” and can be used inside the function.

Attention!

The “internal_variable” exists only inside the function and cannot be called outside:

```
print(internal_variable)
```



```
Traceback (most recent call last):
  File "/Users/saschapatz/PycharmProjects/Vorkurs2020/python_script.py", line 101, in <module>
    print(internal_variable)
NameError: name 'internal_variable' is not defined

Process finished with exit code 1
```

Write a function, that returns a value ...

```
# Declare a simple return function
def return_me(internal_variable):
    return(internal_variable)

# Call the return function
return_me("World")
```

Type and run the code!

Why the function does not print any value to the terminal?

Write a function, that returns a value ...

```
# Declare a simple return function
def return_me(internal_variable):
    return(internal_variable)

# Call the return function
term = return_me("World")

print(term)
```

Type and run the code again!

Do you understand?

Write a function, that returns a value ...

```
# Understanding the function more in detail
def my_function(my_paramerters='my_default_parameter_value'):
    """ Some small description of what this function does """
    print('my_parameters', my_paramerters)
    return 'any return value'

print('Call without parameters')
my_function()

print('Call with parametes')
my_function(my_dictionary)
```

Use Exceptions ...

```
try:
    ## whatever here
    my_function()
except Exception: ## generic exception can be here
    # what to do after it is caught?
    print('Could not perform!')
```

Type and run!

Time to
Try and Error
Q and A!

Conditional Programming and Loops

What is an If Statement?

Conditional Programming

```
# If statement
i="a"
if i == "a":
    print("found a")
else:
    print("not a")
```

If statement:
Intendation and action to do if true.

Optional else statement:
Intendation and action to do if if statement is false.

Intendation: use a tabular space (Tab)

```
# If statement
i="a"
if i == "a":
    print("found a")
else:
    print("not a")
```

Conditional expressions

```
(x<5) and (y>2) or (x==3)
(x<5) and (y>2) and (x==3)
(x<5) or (y>2) and not (x==3)
```

or	not	is	is not
and	<	<=	>
not x	>=	!=	==
in			

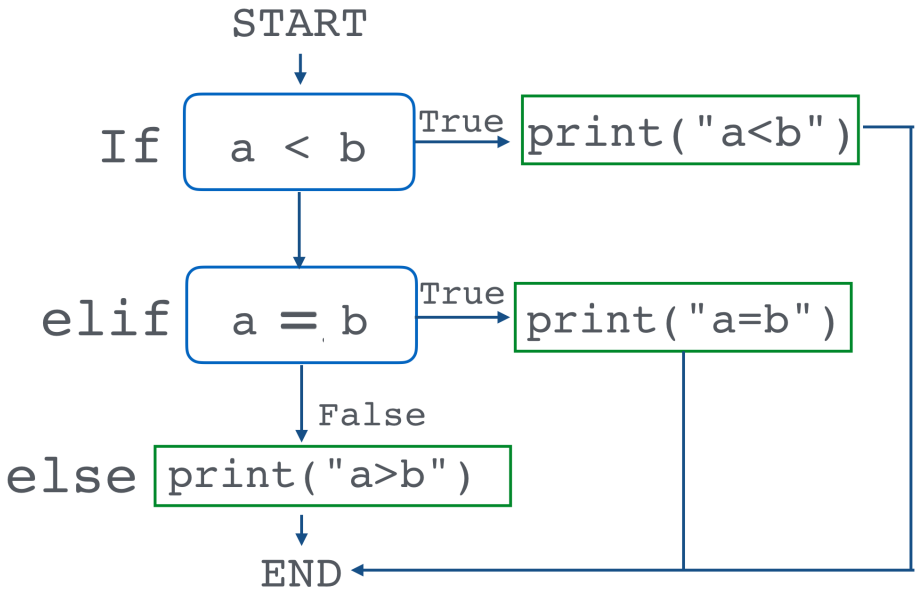
```
# If:
if a:
    print('a is true!')
else:
    print('a is false!')

empty_list = []
if empty_list:
    print('empty_list is True!')
else:
    print('empty_list is False..')

## what does if really do in here?
```

Conditional Programming – Try it!

a = 10
b = 3



Try by yourself to implement the conditions.

Why to pay attention on choosing the appropriate elif statement?

What is a For Loop?

Your first for loop ...

```
# For loops
for i in range(5):
    print(i)
```

Type and run!

Take care of intendation: use a tabular space (Tab)

Look up the arguments of the range() function! Any default values given?

Syntax:

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

https://pynative.com/python-range-function/

What is a While Loop?

Your first while loop ...

```
# While loops
i=0
while i < 10:
    print('i is still smaller then 10!', i)
    i+=1

print('i is 10!', i)
```

ATTENTION:
If you choose the wrong while statement, so that it is always true, the loop may run forever!

Type and run!

Take care of intendation: use a tabular space (Tab)

Useful string operations …

```
my_string = 'Bioinformatics'

# Changing capitalisation
print(my_string.lower())
print(my_string.upper())

# Check for prefix and suffix
print(my_string.startswith('B'))
print(my_string.endswith('B'))

print(my_string.replace('a', 'B'))
print(my_string.split('o'))

# String concatenation:
a = 'Horses'
b = ' and '
c = a + b

# Checking for substring
print('and' in c)

# Iterate over a string like over a list
for letter in my_string:
    print(letter)
```

Can you apply more list operations on a string, try it!

Useful list operations …

```
my_string = 'Bioinformatics'
print(my_string)

my_list = list(my_string) # casting a string to a list
print(my_list)

my_list.append('Dynamic lists!') # add an item to a list
print('A' in my_list)

print('o' in my_list)
print(my_list)

print(len(my_list))

# list to str:
print('-'.join(my_list[:-1])) # slicing: all but not the last element
print('-'.join(my_list[2:-5])) # slicing

# str to list:
print(my_string[1:10])

# sort a list
my_list.sort()
print(my_list)
```

Checking lists ...

```
l = [1, 3, 5, 'Horses']
print(1 in l)
print(1 in [1, 3, 5, 'Horses']) # list can be written instead of variable

a = 'Horses and Goats'
print(a in [1, 3, 5, 'Horses'])

# to be used in an if statement
if a in l:
    print("yes!", a, "found in ", l)
```

WORK ON YOUR OWN !

ZapHep

Write a function, applying if, elif, else, to play Zap Hep!
It prints sequential integers until a maximum value, that is given as argument in form of a variable,
if a number is divisible by 3, print ZAP instead,
if there is a 3 in the number print HEP.
If its both, print ZAP HEP.

```
Output:
1
2
ZAP
HEP
4
5
ZAP
7
8
ZAP
10
11
ZAP
```

```python
# Game Zap Hep
def run_zaphep(a):
    ...             #fill the function

a = 10
run_zaphep(a)
```

List comprehension ...

```
input_par = [1,2,3,4,5,6,7,8,9,10]

# for loop
for i in input_par:
    print(i)

# list comprehension
[print(i) for i in input_par]

# for loop with if
input_par = [1,2,3,4,5,6,7,8,9,10]
b = []
for i in input_par:
    if i % 3 == 0:
        b.append(i)

print(b)

# list comprehension with if
b = []
b = [i for i in input_par if i % 3 == 0]

print(b)
```

WORK ON YOUR OWN !

Write a function to:

1. count **“a”** in a given sequence.
2. count **a**,**t**,**g** in a given sequence using a dictionary.
3. count any letter in a given sequence using a dictionary
4. create a reverse complement sequence

```
sequence = “atgaagattc”
```