

Python Programming for Life Scientists

Hanna M. Schilbert, Katharina Sielemann, Bianca Frommer, Daniela Holtgräwe

Genetics & Genomics of Plants and Computational Biology,
CeBiTec, Bielefeld University



Reminder: 'Studienleistungen' – study requirements

- **Active participation**
- **Documentation of course exercises**
- **Python project**
 - Write a script
 - Give a presentation about it (10-15min)
 - Deadline for the submission of the python script is **07.04.2022 (1 pm)**
 - Presentations will be given on the **08.04.2022 (10 am), online via Zoom**

Group mailing list

- `hschilbe@cebitec.uni-bielefeld.de`
- `ksielemann@cebitec.uni-bielefeld.de`
- `frommer@cebitec.uni-bielefeld.de`
- `dholtgra@cebitec.uni-bielefeld.de`

Research



Gene prediction

Functional annotation



De novo genome assembly

De novo transcriptome assembly

Non-canonical splice sites

Genome Research

RNA-Seq analysis

Synteny

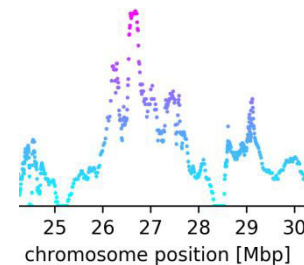
Co-expression analysis

Phylogeny

Gene families / pathways



Mapping-by-sequencing
(MBS)



Documentation

- Please make notes throughout the whole course
- At the end of each day, sum up what you did
- ...

Aims & content

- Learning the basics of Python syntax



```
def print_hello_world():  
    """function prints 'hello world!'"""  
    print("hello world!")  
print_hello_world() #calling function
```

```
hello world!
```

Aims & content

- Learning the basics of Python syntax
- Parsing, filtering, exporting, converting

```

ncRNA              :: 1
Frameshifted Genes :: 35
##Genome-Annotation-Data-END##
COMPLETENESS: full length.
FEATURES             Location/Qualifiers
     source            1..5879062
                        /organism="Xanthomonas campestris pv. campestris"
                        /mol_type="genomic DNA"
                        /strain="B100"
                        /db_xref="taxon:340"
                        /pathovar="campestris"
     gene              1..1329
                        /locus_tag="XCCB100_R500005"
                        /old_locus_tag="xcc-b100_0001"
                        /old_locus_tag="xccb100_0001"
     CDS               1..1329
                        /locus_tag="XCCB100_R500005"
                        /old_locus_tag="xcc-b100_0001"
                        /old_locus_tag="xccb100_0001"
                        /inference="EXISTENCE: similar to AA
                        sequence:SwissProt:Q8PRG2.1"
                        /note="Derived by automated computational analysis using
                        gene prediction method: Protein Homology."
                        /codon_start=1
                        /transl_table=11
                        /product="chromosomal replication initiator protein DnaA"
                        /protein_id="WP_011035259.1"
                        /db_xref="GI:499345720"
                        /translation="MDAWPRCLERLEAEFPEDVHTLKLPLQAEDRGDSIVLYAPNAF
                        IVEQVRERYLPRIRELLAYFAGNGEVALAVGSRPRAPEPLPAQAVASAPAAAPVVF
                        AGNLDSHYTFANFVEGRSNQLGLAAAIQAAQPGDRAHNPPLLGGTGLGKTHLMFAA
                        GNALQANPAKMYLRSQEFSSAMIRALQKAMDQPKRQFQIDALLIDDIQFFAGK
                        DRTQEEFFHTFNALFDGRQIIITCDVPREVLEGLPRLKSLAAGLSVAIDPPDET
                        RAATVLAKARERGAEPDQVAFILAKKMRNSVRDLLEGALNTLVARANFTGRSITVEFA
                        QETLRDLLRAQQQAGIPNIQKTADYVYGLQMKDLSKRRTSRLARPRQVAMALAKEL
                        TEHSLPEIGDAFGRDHTVLHACRQIRTLEADGKLRDWEKLRKLSLSE"
     gene              1605..2705
                        /locus_tag="XCCB100_R500010"
                        /old_locus_tag="xcc-b100_0002"
                        /old_locus_tag="xccb100_0002"
                        /old_locus_tag="xccb100_0002"
     CDS               1605..2705
                        /locus_tag="XCCB100_R500010"
                        /old_locus_tag="xcc-b100_0002"
                        /old_locus_tag="xccb100_0002"
                        /old_locus_tag="xccb100_0002"
                        /EC_number="2.7.7.7"
                        /inference="EXISTENCE: similar to AA
                        sequence:RefSeq:WP_006451791.1"
                        /note="binds the polymerase to DNA and acts as a sliding
                        clamp; Derived by automated computational analysis using
                        gene prediction method: Protein Homology."
                        /codon_start=1
                        /transl_table=11

```



```

>xccb100_0001  chromosomal replication initiation protein
MDAWPRCLERLEAEFPEDVHTLKLPLQAEDRGDSIVLYAPNAFIVEQVRERYLPRIRE
LAYFAGNGEVALAVGSRPRAPEPLPAQAVASAPAAAPVVFAGNLDSHYTFANFVEGRS
NQLGLAAAIQAAQPGDRAHNPPLLGGTGLGKTHLMFAAGNALRQANPAKMYLRSQEF
FFSAMIRALQKAMDQPKRQFQIDALLIDDIQFFAGKORTQEEFFHTFNALFDGRQII
LTCDVPREVLEGLPRLKSLAAGLSVAIDPPDETFAATVLAKARERGAEPDQVAFIL
AKKMRNSVRDLLEGALNTLVARANFTGRSITVEFAQETLRDLLRAQQQAGIPNIQKTAD
YYGLQMKDLSKRRTSRLARPRQVAMALAKELTEHSLPEIGDAFGRDHTVLHACRQIR
TLMEADGKLRDWEKLRKLSLSE*
>xccb100_0002  DNA polymerase III subunit beta
MRFTLQREAFKPLAQVNVVVERQTLPVLANLLVQVNNQSLTGTGLEVMISRTMVE
DAQDGETTIPARKLFDILRALPDGSRVTVSQDQVTVQAGRSRFTLATLPANDFPVSVE
VEATERVAVPEAGLKELMERTAFAMAQQQVRYVLLNCLFDRLDGLRCVATDGHRLALCE
TELEKSGSAGKRIIVPRKGVTELLRLLEAADRDVELELGRSHIRVKRGDVFSTKLIDGR
FPDYEAIVPIGADREVKVDREALRASLQRAAILSNEKYRGVRVVEVSGQLKISAHNPEQE
EAQEEIEADTKVDDLAIGFNWVYLLDALSLRDEHVVIQLRDANSALVREASSEKSRHV
VMPLRL*
>xccb100_0003  recombination protein F
NSTADHVCSAPSDGLCGQADRSMHVARLSHRLRFAVEFHPASTNLLTGNGACGKT
SVLEALHVMAYGRSFRGVRDGLTQGGQDLETFVEWRERAGDSTERTRAGLRHSGQEW
TGRLDGEDVQAQGLCSALAVVTFEPGSHVLSGGGEPRRRLDNLGFHVEPDFLALWRR
YARALKQRNALLKQGAQPMQDANDHELAESGETLTSRLLQYLERLQERLVPVATAIAPS
LGLSALTFFAPGRHREVSADALLARERDRNGVTSQPHRADWAPLFDALPGKDALSR
GQAKLTALACLLAQAEQFAHERGEWPMALDDLSGSELDHRHQARVQIRLASAPAVLITA
TELPPLGADAGKTLRFHVEHGLVLPQPLTPDPPRLA*
>xccb100_0004  DNA gyrase subunit B
MTDEQTPPTPTNGTVDSKSTIVLRLGLEAVRKRPPCHYIGDHDGTLGHMMFVEVDNSVE
ALAGHADDIVKIHVDGSAVSDNGRGVVDIHKEGVSAAEVILTVLHAGGKFDNSYK
VSGGLHGVGVSVVNALSEHLWLDIWRDGHYQYEALEGPQYPLKLEASTKRGTTLRFK
PAVEIFSDVEFHVDILARRLRELSLNSGVKIALIDERGEGRRDHFYEGGIRSFVEHLA
QLKTPLHPNVISVTGEHNGIVVDVALQWTDAYQETHYCTNNIPQDKGGTHLAGFGALT
RVLNRYIEQNGIAKQAKITLTDGDMREGMAIVLSVKVPDPSFSQTKERLVSSDVRPAVE
NAGADQETFLQENPMEATGATIKTLDAAARAEAAKARDLTRKGLDIAQLPKLADQ
QEKDPALSELFTVEGDSAGSAGKGRNKNQAVLPLRCKILNVERARFDRMLASQVDTGL
ITALGTGIGRDEYNPKRLRYHRIILMTADVDGSHIRTLTLFFYRQMPELIERGYIYIG
LPLLYKLGKQKSELYLKDAAANLAYLASSAVEGAALIPASDEPPIITGEALKLLFAGA
KEAIARNAHRYDPALLTALIDLPPLDVVQLQAGDVHPTLDALQAVNRLGTLTARYHLR
FDPATDSAAASLVSRKHMGEEFTQVLPMAFESGELRPLREVALALHGLVREGAQLIRG
NKSHPTISFAQADQWLEEAQRGRQVRQFKGLGEHNAEQLEMTTVNPDTRRLQVRTEDA
VAADQIFSTLKGDVPRFEDINALKYSNLDI*
>xccb100_0005  putative membrane protease
MSAVLPSPAPVSPGPPSLRSVAVLFCIDLLTAIGLLLSLVAGFAVNGFLRSMGEVQA
VRAQGSQSPAPAIATGQPGVNVQLLIALVSTATPAVLLYFHRRRATPAEQATSRRAIR
RRTTGWNTAAVAGVFMLSNLVSLASALGPKVPTNLPLMEAIKQMLALVVFVAIA
PAYEELLFRVRLFGRLAAGRPHLGVLSSLTALVHEVPGISGNGVATAGLVLVGGH
GAFAHLYHRTCTLWAPTLAHGIIINNALALVFFGLG*
>xccb100_0006  putative exported peptidase/protease
MKVRLILVAVLALCATATTTSPTGRQVGGVTDQDLKLGAESFAQTKAKEKVSITDGK
QNAYVQCVVNALVAQLPPQWRETRWETALFVDEANAFALPGKGVGNTGIFTVAKTQDQ
LAVALGHEIGHVIRHHEERTLQGLAQTLGTIGLAGAAGYGDGAASVNVQVGMGTAGT
VFLLPGRSTQSEADVQGRMAQAGFDPAQAVSLWQNMMAASGNRQWLSLTHPDANR
IRELQADVNALQVYQARQDGRVPRCG*

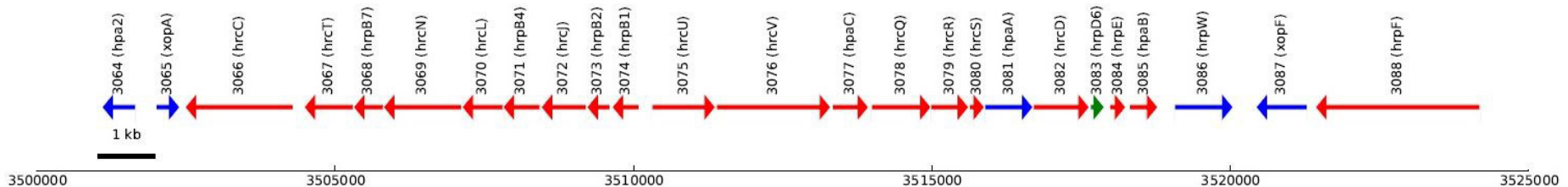
```

Genbank (.gb)

FASTA (.fa/.fas/.fasta)

Aims & content

- Learning the basics of Python syntax
- Parsing, filtering, exporting, converting
- Generate figures (e.g. matplotlib)



Aims & content

- Learning the basics of Python syntax
- Parsing, filtering, exporting, converting
- Generate figures (e.g. matplotlib)
- Statistics for biological data



Aims & content

- Learning the basics of Python syntax
- Parsing, filtering, exporting, converting
- Generate figures (e.g. matplotlib)
- Statistics for biological data
- **Answer biological questions!**



Aims & content

- Learning the basics of Python syntax
- Parsing, filtering, exporting, converting
- Generate figures (e.g. matplotlib)
- Statistics for biological data
- **Answer biological questions!**
- Own projects/challenges?! => own solution & success



Why Python?

- Easy to learn
- More efficient than Excel => important for big data
- Check/modify => unlimited opportunities
- Script needs to be written once and can be applied often
=> Frequent problems are ideal for bioinformatics
- Very powerful with many libraries/modules
=> Biopython (including NCBI BLAST+), scipy, numpy, Rpy, matplotlib, scikit-learn, ...



Examples

- Everything Excel does and much more
- Primer design and validation
 - Identify binding sites and their distances, nucleotide composition, codon frequency, etc.
- BLAST + evaluation of results
 - *in silico* translation of multiple sequences and automatic inspection of all resulting gene products
- Convert data formats (e.g. FASTA-like > FASTA)
- Advanced search & replace
- Your own ideas!!!

Outline

- Environment: Jupyter notebooks, Python
- Basic commands and data structures
- Functions
- Control structures (if, else, for, while)
- File handling
- Statistics
- Solving biological questions with real data
- ...
- Your own project!!!

Environment: Jupyter notebooks & Colab

Jupyter notebooks & Colab

- Google account (+ PW) required

What is Colaboratory?

Colaboratory, or "Colab" for short, allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier.

Code blocks and text blocks

▼ Getting started

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
[ ] seconds_in_a_day = 24 * 60 * 60
    seconds_in_a_day
```

```
86400
```

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut "Command/Ctrl+Enter". To edit the code, just click the cell and start editing.

Variables that you define in one cell can later be used in other cells:

```
[ ] seconds_in_a_week = 7 * seconds_in_a_day
    seconds_in_a_week
```

```
604800
```

Libraries

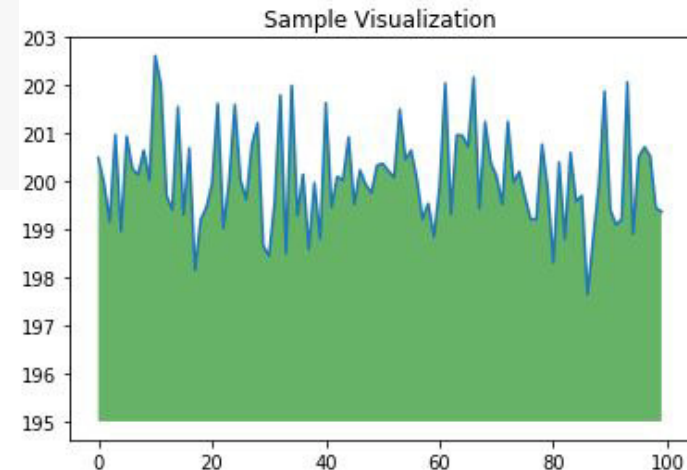
- Combine **executable code** and **rich text** in a single document, along with **images**, **HTML**, **LaTeX** and more
- Use Python libraries (e.g. matplotlib)

```
[ ] import numpy as np
    from matplotlib import pyplot as plt

    ys = 200 + np.random.randn(100)
    x = [x for x in range(len(ys))]

    plt.plot(x, ys, '-')
    plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)


    plt.title("Sample Visualization")
    plt.show()
```



Introduction to Colab

Get started with Google Colaboratory

<https://www.youtube.com/watch?v=inN8seMm7UI>



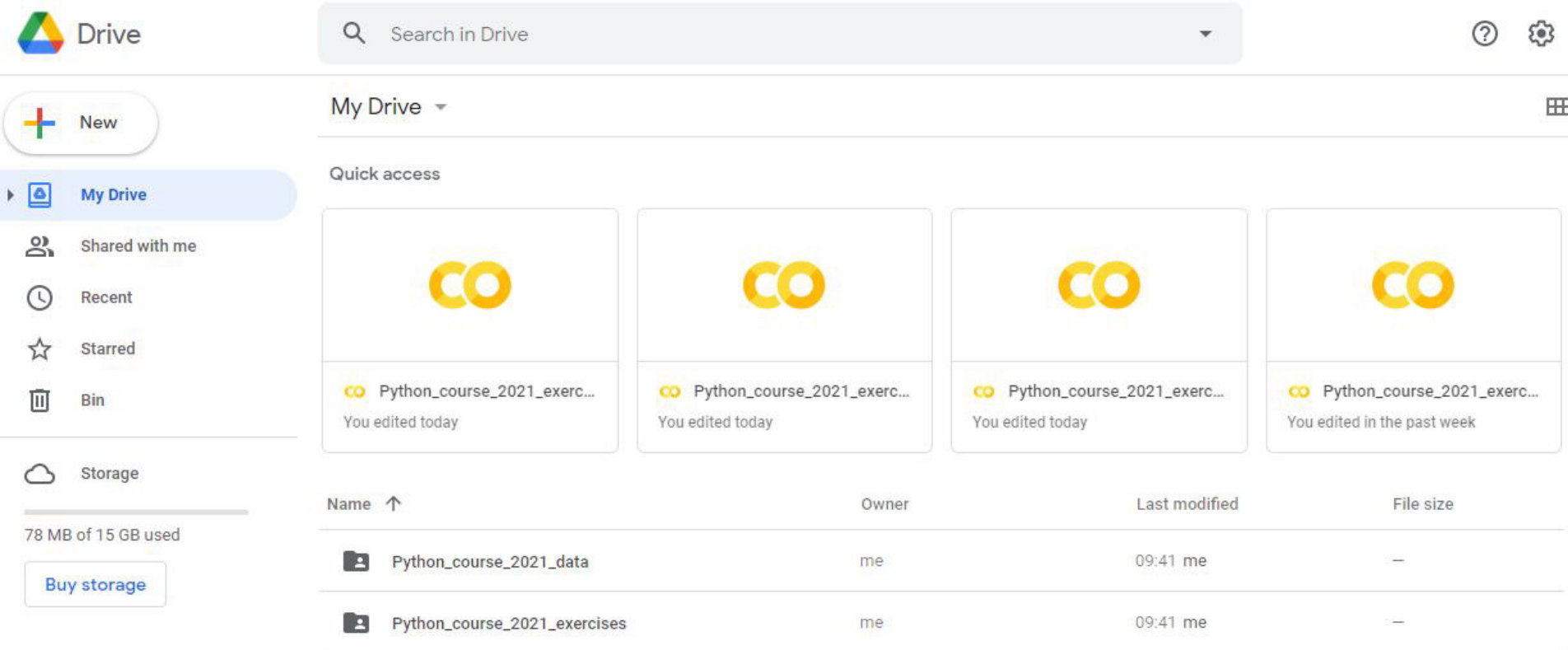
What is Colaboratory?

Colaboratory, or "Colab" for short, allows you to write and execute Python in your browser, with



- Zero configuration required
- Free access to GPUs
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier.

File manager – Google Drive

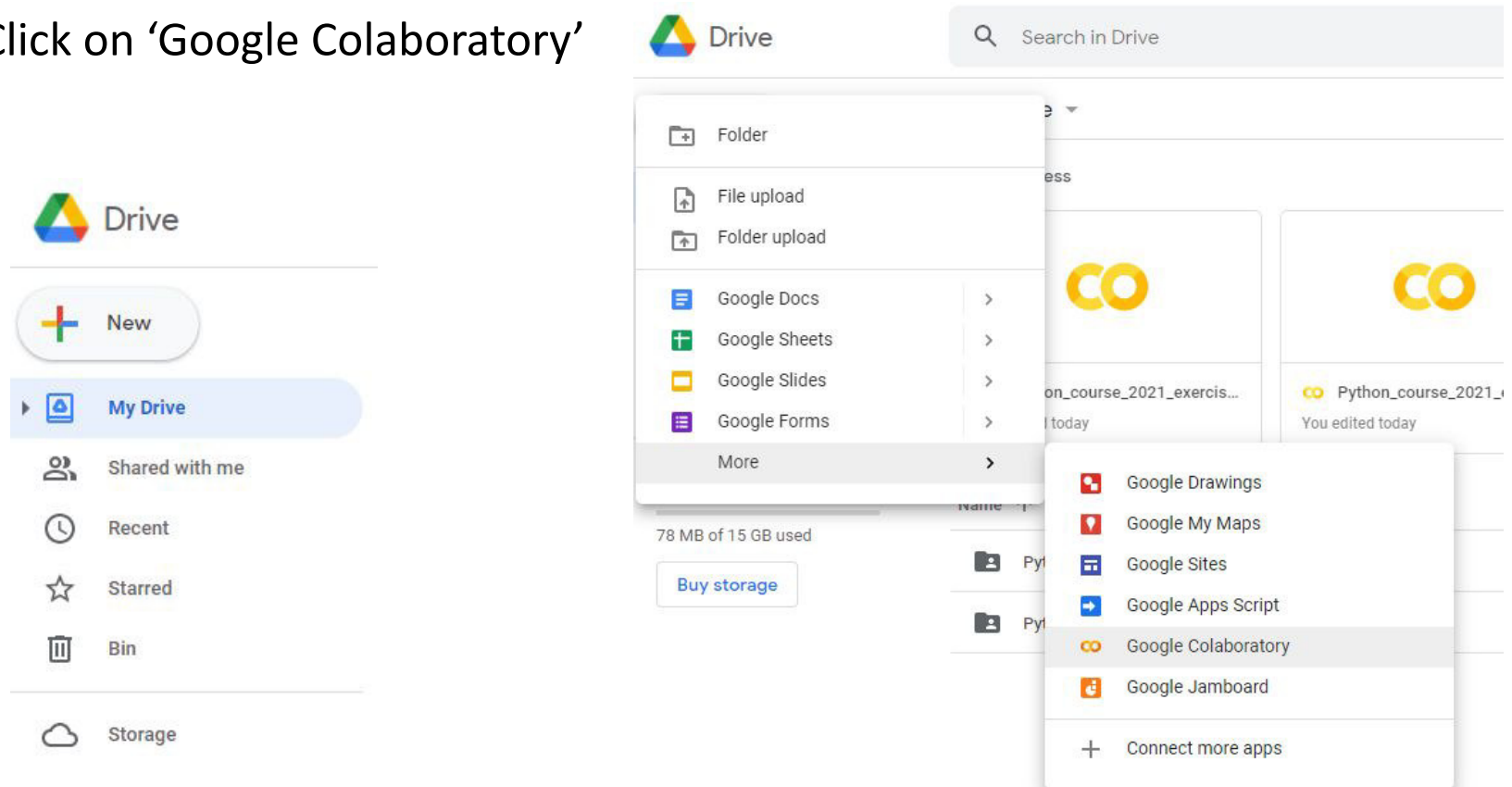


The screenshot shows the Google Drive web interface. On the left is a sidebar with navigation options: 'New' (plus icon), 'My Drive' (selected), 'Shared with me', 'Recent', 'Starred', 'Bin', and 'Storage' (cloud icon). The main area is titled 'My Drive' and features a 'Quick access' section with four file thumbnails, each showing a yellow 'CO' logo and the filename 'Python_course_2021_exerc...'. Below this is a table of files.

Name ↑	Owner	Last modified	File size
 Python_course_2021_data	me	09:41 me	—
 Python_course_2021_exercises	me	09:41 me	—

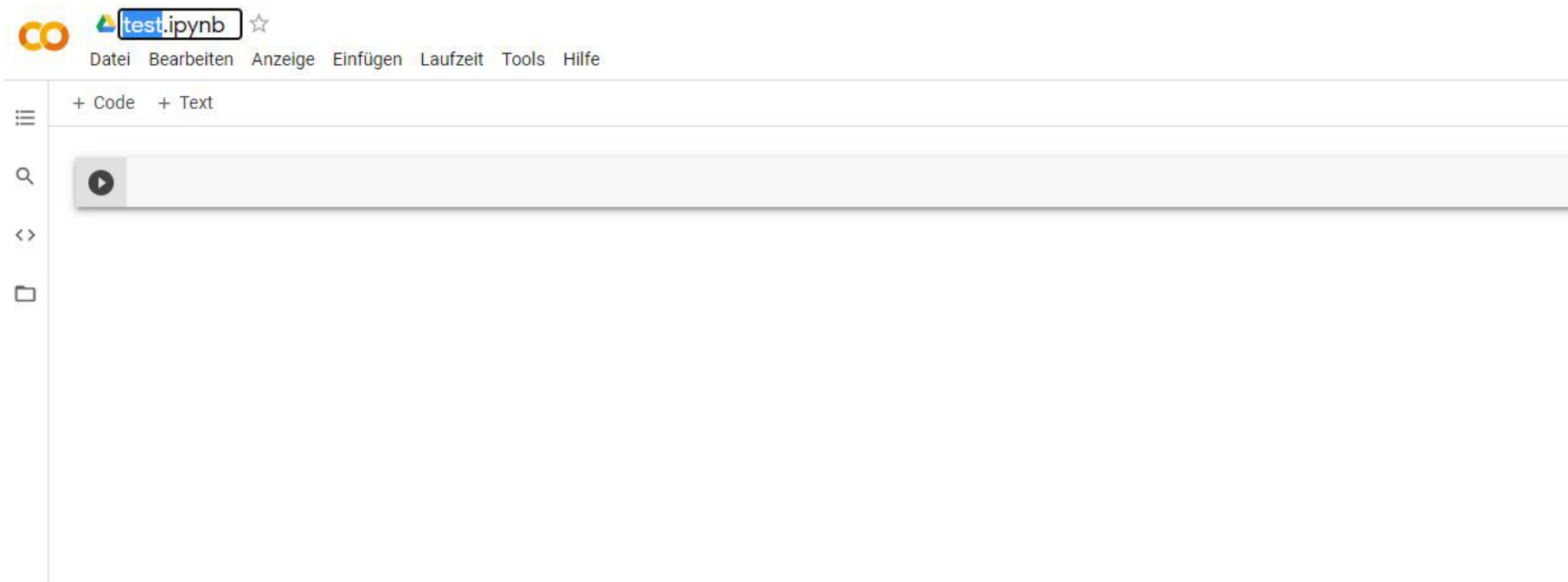
LET'S START - The first Python script

- Create new colab file by clicking on the '+' button
- Choose 'more'
- Click on 'Google Colaboratory'



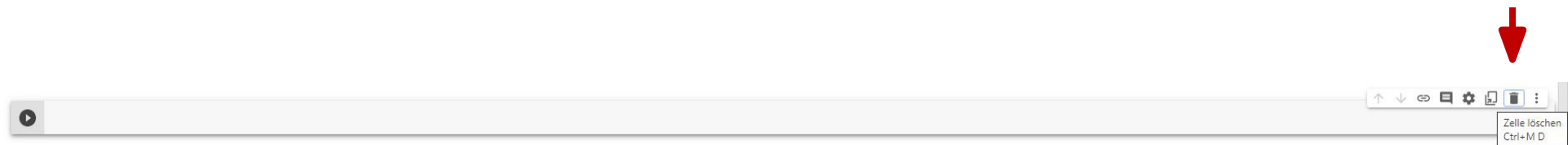
LET'S START - The first Python script

- Name your file: 'test'



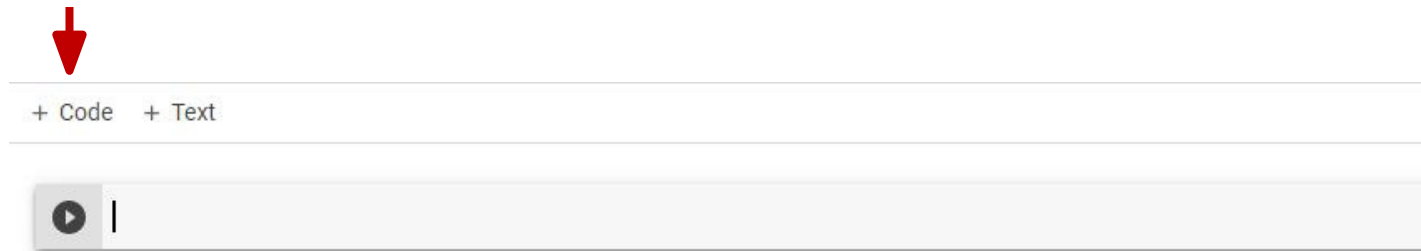
LET'S START - The first Python script

- Delete the existing code block



LET'S START - The first Python script

- Add a code block
- Add a comment using `#text` or `“text”` and write ‘this is my first Python script’
- Write `print(“hello!”)`




LET'S START - The first Python script

- Run your first script

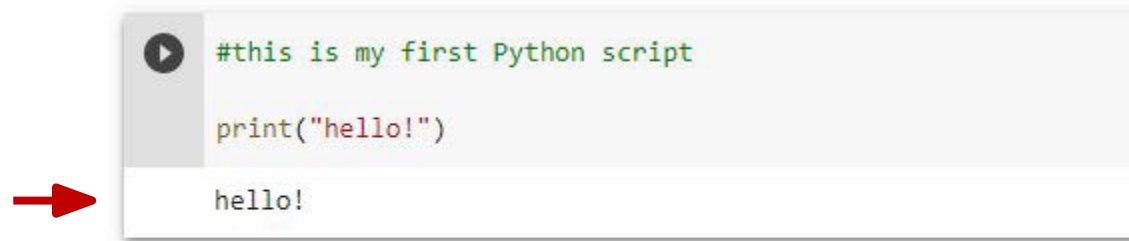
+ Code + Text

```
#this is my first Python script  
print("hello!")
```



LET'S START - The first Python script

- Output is directly attached to your code block



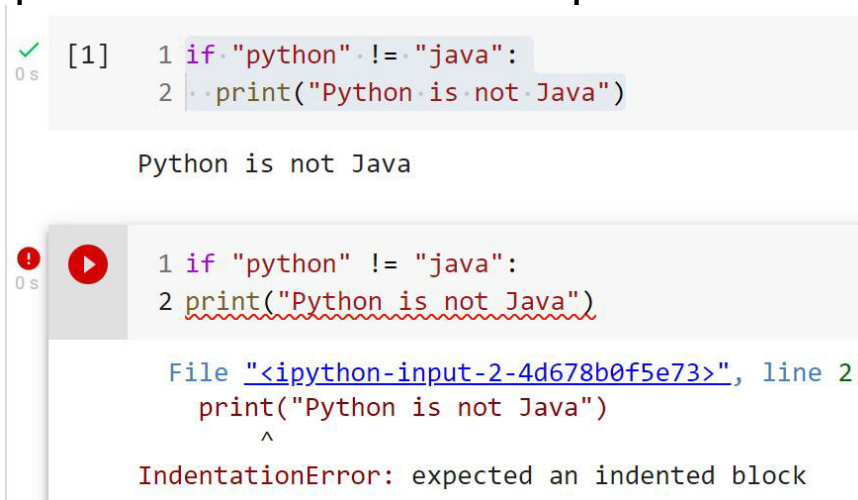
```
#this is my first Python script  
print("hello!")  
  
hello!
```

Data types & Variables

```
1 str()                #Text type
2 int(), float(), complex() #Numeric types
3 list(), tuple(), range() #Sequence types
4 dict()               #Mapping type
5 bool()              #Boolean type
6
7
8 # examples:
9 print('test', "test") #command line output
10 my_int = 1            #integer
11 my_float = float(3.12154) #float
12 my_string = "Hello!"  #string
13 my_string2 = str( my_int ) #number to string
14 my_list = ['1', '2', '3', 4, 5, 6] #list
15 my_dict = {'apple' : 'banana', 'fruit': 'pineapple'} #dictionary
```

Structure

- Use ASCII characters only (no ä, ö, ü, ß ...)
- Empty lines are ignored by Python
=> Use space to structure code
- Be careful with (regular expression) signs: |, \$, %, !, =
 - ! They have specific functions
- No space in names of variables, functions
- Structure is important: indentation is required inside a code block



The image shows two examples of Python code execution. The top example shows a correct code block with two lines of code: `if "python" != "java":` and `print("Python is not Java")`. The code is executed successfully, and the output is "Python is not Java". The bottom example shows the same code block, but the second line is not indented. This results in an `IndentationError` with the message "expected an indented block". The error message points to the second line of code.

```
[1] 1 if "python" != "java":
    2 print("Python is not Java")


Python is not Java
```

```
1 if "python" != "java":
2 print("Python is not Java")

File "<ipython-input-2-4d678b0f5e73>", line 2
    print("Python is not Java")
    ^
IndentationError: expected an indented block
```

Variables I

- Create a variable with name 'a'



```
1 a = 'Hello world'
2 print(a)
```

Hello world

- Both single ('a') and double ("a") quotes are allowed
- Variable names
 - Must start with a letter or underscore
 - Only alphanumeric characters plus underscore allowed (A-z, 0-9, _)
- Use meaningful variable names (e.g. list_of_strings)

Variables II

- Multiple values & multiple variables:

```
1 a, b, c = 'Hello', 'world!', 'How are you?'  
2 d = e = f = "Hi!"  
3 print( a, b, c)  
4 print( a + b + c)  
5 print(d, e, f)
```

```
Hello world! How are you?  
Helloworld!How are you?  
Hi! Hi! Hi!
```

Comments

- Why comments?
 - Explains your Python code => Increases the re-usability of your code
 - Others (including yourself after a while) can easily understand the code
 - Prevents some code lines to be executed
- One-line comments: '#'

```
[3] 1 # Hi! I am a comment. How are you?  
    2 print("Hello world")
```

Hello world

```
[4] 1 print("Hello world") # Hi! I am a comment. How are you?
```

Hello world

```
[5] 1 # print("Hello world")  
    2 print("Hi! How are you?")
```

Hi! How are you?

Comments

- Multi-line comments: """ comment """



```
1 # multi
2 # line
3 # comment
4 print("Hello world")
```

Hello world

```
[7] 1 """
     2 multi line comment
     3 """
     4 print("Hello world")
```

Hello world

Assignment / comparison

- '=' used to assign value to variable:

```
1 a = "Hello world,"  
2 b = 'how are you?'  
3 c = 10  
4 print(a, b, c)
```

Hello world, how are you? 10

- '==' compares two values/variables:

```
1 d = 5  
2 e = 5  
3  
4 if d == e:  
5 |     print(d+e)
```

10

- Variable names may contain characters, underline, and numbers (not at the start!)

Variable type string

- a, b are strings ("str")
- Python allows to check the variable type:

```
1 a = "Hello world"  
2 type(a)
```

str

- Almost all variable types can be converted to string:

```
1 c = 10  
2  
3 print(type(c))  
4 print(type(str(c)))
```

```
↳ <class 'int'>  
   <class 'str'>
```

Variable types integer & float

- Two variable types for numbers:
 - integer = complete number (example: 3)
 - float = decimal number (example: 3.1415926)
- Important: “.” NOT “,” separates numbers in float! (English syntax)
- Some strings can be converted to integer/float:

π

```
1 my_int = int("3")
2 my_float = float("3.145926")
3
4 print(type(my_int), type(my_float))
```

```
<class 'int'> <class 'float'>
```

Python as calculator

- Numbers can be used for calculations ;-)



```
1 a = 3
2 b = 2
3 print(a+b)      #addition
4 print(a*b)      #multiplication
5 print(a**b)     #exponentiation
6 print(a/b)      #division
7 print(a/float(b))
8 print(a%b)      #modulo division
9 print(a < b )    #alternatives: >=, <=, >, and ==
10 print(a != b)  #test for inequality
```

```
5
6
9
1.5
1.5
1
False
True
```

- Calculating roots?
- Interested in more complex math? => numpy, scipy

Variable type list

- List can contain elements of different types:

```
1 list_str = [ "one", "two", "three" ]  
2 list_int = [ 1, 2, 3 ]  
3 list_list = [['List'], ['in', 'list']]  
4 print('List of strings: ', list_str)  
5 print('List of integers:', list_int)  
6 print('List of lists:  ', list_list)
```

```
List of strings: ['one', 'two', 'three']  
List of integers: [1, 2, 3]  
List of lists:  [['List'], ['in', 'list']]
```

- Elements can be accessed via index

```
1 list_list = [['List'], ['in', 'list']]  
2 print('First element of list:', list_list[0])
```

- Matching your expectation?

Indices in Python

- Python starts counting at 0!!!

```
1 list_str = [ "one", "two", "three" ]  
2 # Index      0      1      2  
3 print('Elements:', list_str[0], list_str[1], list_str[2])
```

Elements: one two three

- Lists can be concatenated:

```
1 list_int = [ 1, 2, 3 ]  
2 new_list = list_int + [ 'four', 'five', 'six', 'seven' ]  
3 print(new_list)
```

[1, 2, 3, 'four', 'five', 'six', 'seven']

Indices in Python II

- Print subset of list:

```
1 new_list = [ 1, 2, 3, 'four', 'five', 'six', 'seven' ]  
2 print(new_list[ 3: ])      #elements with index to end  
3 print(new_list[ :3 ])      #elements in front of the given index  
4 print(new_list[ 3:5 ])      #elements with first index in front of second  
5 print(new_list[ -1 ])      #last element  
6 print(new_list[ -2 ])      #second last element
```

```
↳ ['four', 'five', 'six', 'seven']  
   [1, 2, 3]  
   ['four', 'five']  
   seven  
   six
```

- Strings have indices as well:

```
1 a = 'hello world test string!'  
2 print(a[0:])  
3 print(a[6:11])  
4 print(a[-1])    #last element  
5 print(a[-7:-1]) #starting from 7th last element to last one
```

```
hello world test string!  
world  
!  
string
```

Variable type boolean (True/False)

- Already used for comparison:

```
1 print(1 == 1)
2 print(1 >= 1, 1 > 1)
3 print(0 == False, 1 == True)
4 print(True + True)
5 print(True + False)
6 a, b = [1], []
7 print( bool(a), bool(b) )
```

```
True
True False
True True
2
1
True False
```

- Boolean variables can be used for calculations (like numbers)
 - False is equivalent to 0, True to 1
- Most of the time used only for internal calculations

Brackets

- Two important types of brackets:
 - [] to generate lists and to access elements via index
 - { } for dictionaries
 - () to transfer arguments to **functions**

=> What are functions?

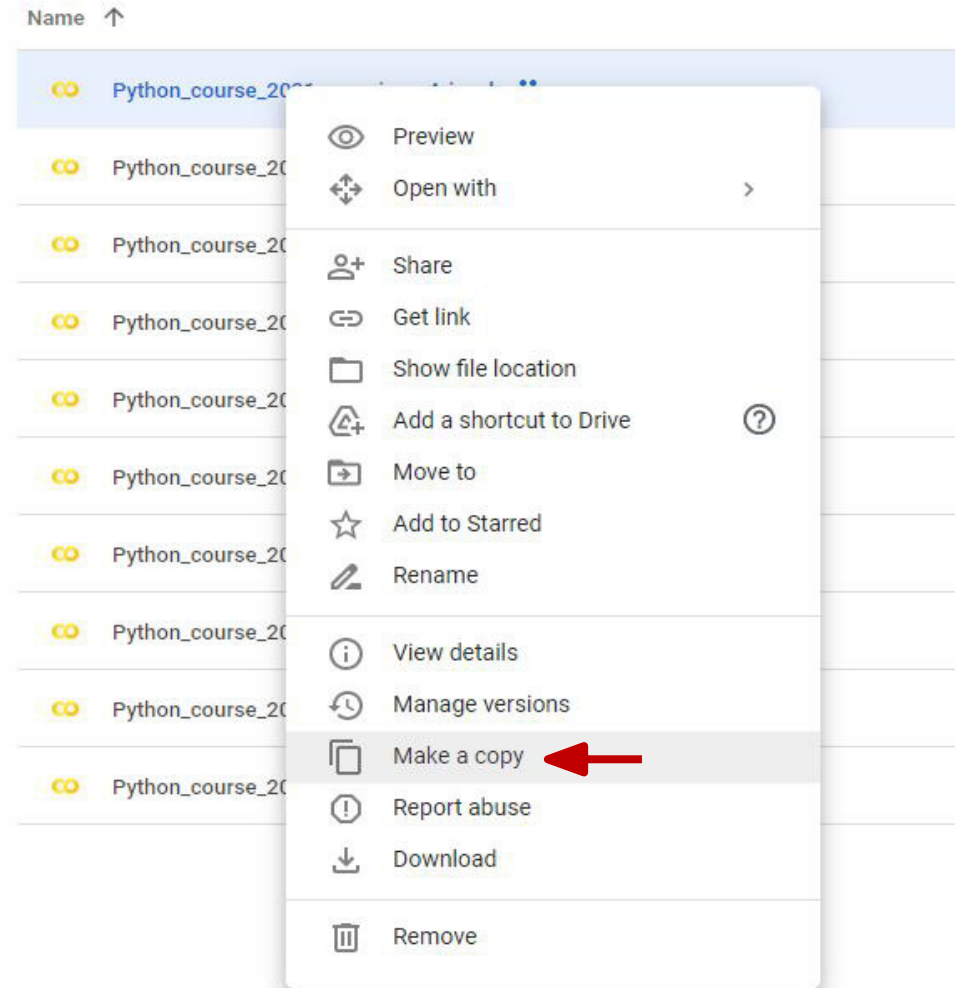
Examples:

```
1 print(str(1))  
2 print(int(1))  
3 print(float(1))  
4 print(len(str(1255)))
```

```
1  
1  
1.0  
4
```

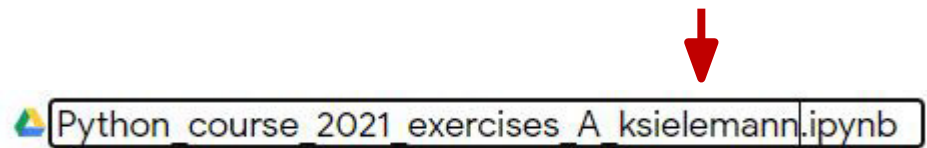
Exercises

- Use the pre-prepared exercise sheets!
- Use the **link** in the chat
- Google Drive should open
- Very **important!**: ,right click' on the file and make a copy!
- Google Colab should open



Exercises

- Rename the file (_yourname)
- Save your own file



Gap Exercises 1

- Access GitLab exercise folder
- Upload 'gap_exercise1_basics.ipynb' into Google Colab
- Fill in/correct the code

Exercises A – Part1

- 1.1) Save 3,14159265359 in a variable of type float!
- 1.2) Convert variable from float to integer!
- 1.3) Convert variable back! What happens?
- 1.4) Convert variable type to string!
- 1.5) Save 'Python' in a string variable!
- 1.6) Convert variable type to float! What happens?
- 1.7) What is a pitfall in regards to division when working with int/float?