

Introduction to



Part 1

Ask!

The art and science of asking questions is the source of all knowledge.

- Thomas Berger

- Do not hesitate to ask!
- If something is not clear, stop me and ask.
- During exercises (you can also ask others).



Failure

- Coding is all about trial and error.
- Don't be afraid of it.
- Error messages aren't scary, they are useful.



[Python natalensis by A. Smith](#) on Wikimedia Commons

History

- Started by Guido Van Rossum as a hobby
- Now widely spread
- Open Source! Free!
- Versatile



Worldwide, Jul 2024 :

Rank	Change	Language	Share	1-year trend
1		Python	29.35 %	+1.5 %
2		Java	15.6 %	-0.2 %
3		JavaScript	8.49 %	-0.8 %
4		C#	6.9 %	+0.1 %
5		C/C++	6.37 %	-0.1 %
6	↑	R	4.73 %	+0.3 %
7	↓	PHP	4.49 %	-0.5 %
8		TypeScript	2.96 %	-0.1 %
9		Swift	2.78 %	+0.2 %
10	↑	Rust	2.55 %	+0.4 %

[PYPL Popularity of Programming Language index](#)

Python today

- Developed a large & active scientific computing & data analysis community
- Now one of the most important languages for
 - Data science
 - Machine learning
 - General software development
- Packages: NumPy, pandas, matplotlib, SciPy, scikit-learn, statsmodels
- Demo: [Link](#)

2 Modes

1. IPython

Python can be run interactively

Used extensively in research

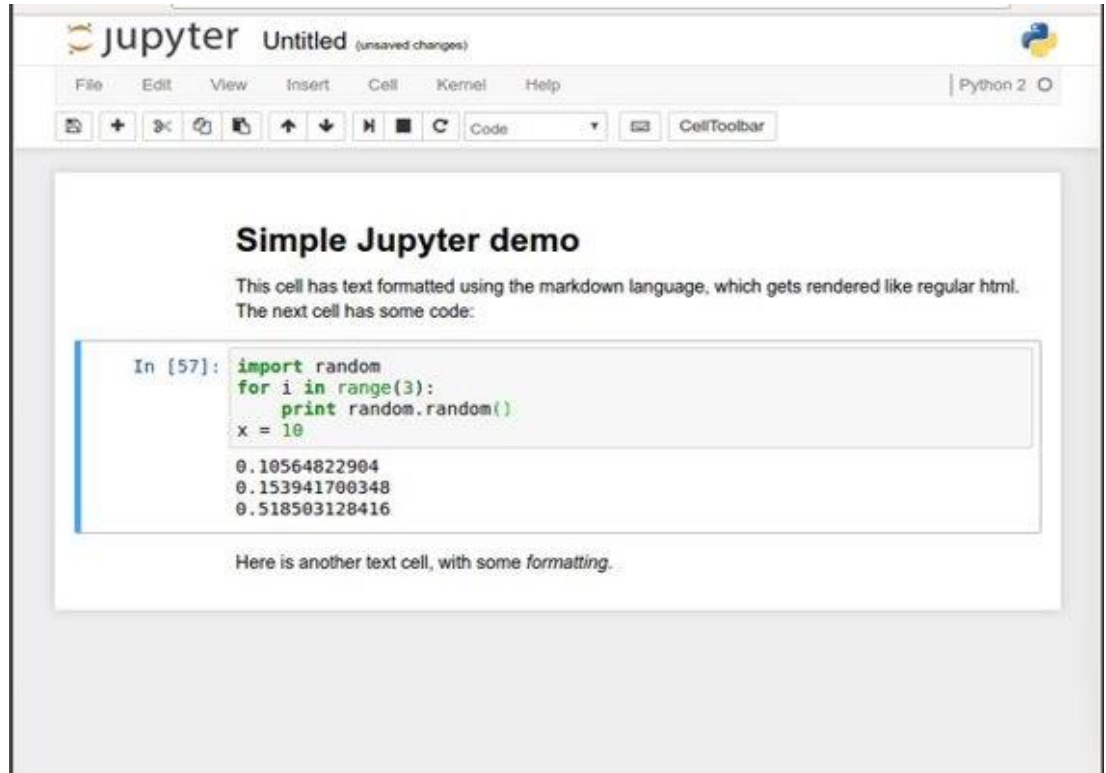
2. Python scripts

What if we want to run more than a few lines of code?

Then we must write text files in .py

Jupyter notebook

- Easy to use environment
- Web-based
- Combines both text and code into one
- Come with a great number of useful packages




Google Colab



Welcome To Colaboratory

File Edit View Insert Runtime Tools Help

 Share



Sign in

+ Code + Text  Copy to Drive

Connect ▾

 Editing



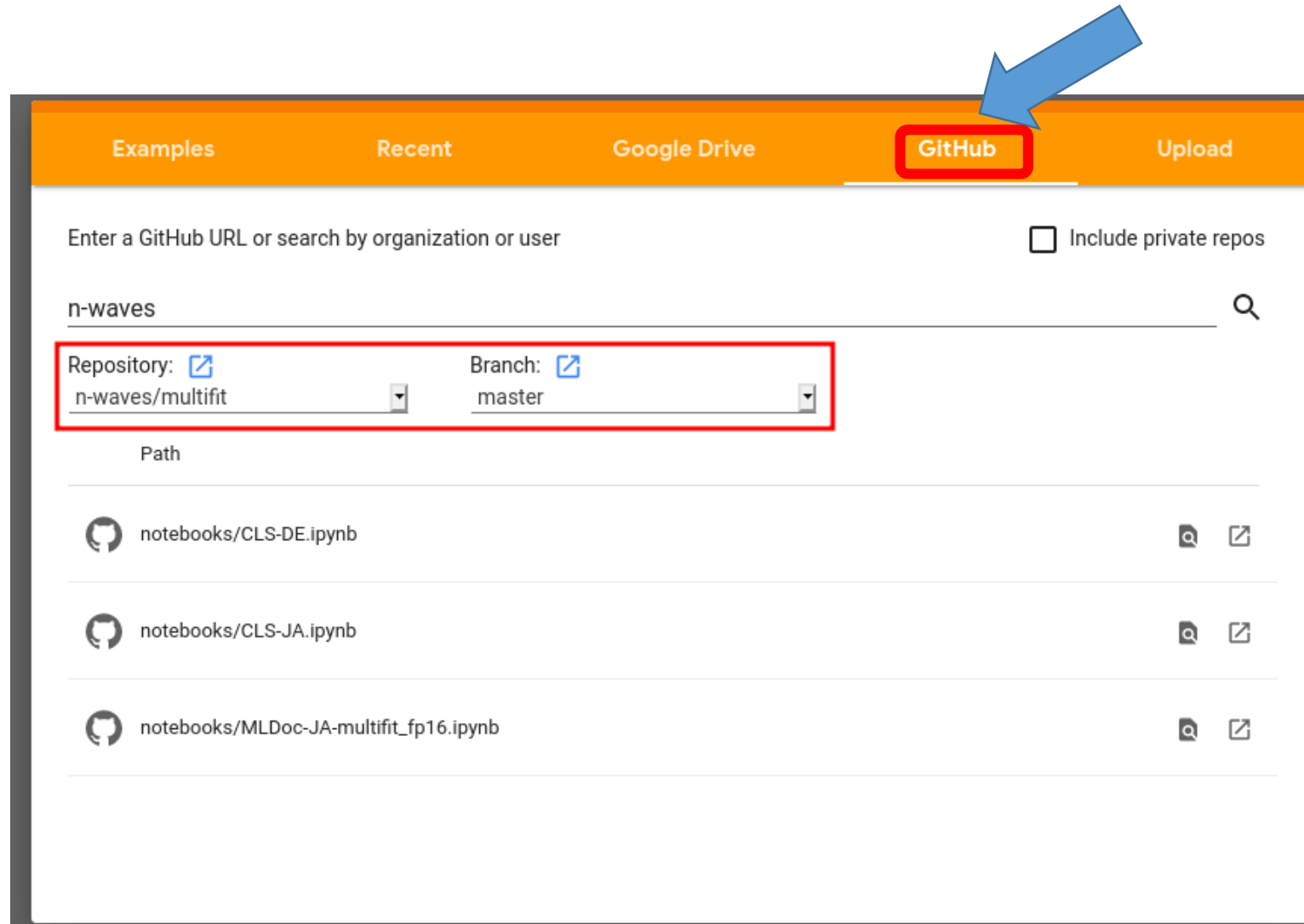
What is Colaboratory?

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. Watch [Introduction to Colab](#) to learn more, or just get started below!

2. Clone GitRepo(recommended)



The screenshot shows the GitHub web interface. A blue arrow points to the 'GitHub' tab in the top navigation bar, which is also highlighted with a red rectangle. Below the navigation bar, there is a search bar with the placeholder text 'Enter a GitHub URL or search by organization or user'. To the right of the search bar is a checkbox labeled 'Include private repos'. Below the search bar, the text 'n-waves' is entered. A red rectangle highlights the 'Repository:' and 'Branch:' dropdown menus. The 'Repository:' dropdown is set to 'n-waves/multifit' and the 'Branch:' dropdown is set to 'master'. Below these dropdowns, the word 'Path' is visible. At the bottom of the interface, there is a list of three notebooks: 'notebooks/CLS-DE.ipynb', 'notebooks/CLS-JA.ipynb', and 'notebooks/MLDoc-JA-multifit_fp16.ipynb'. Each notebook entry has a GitHub icon on the left and a magnifying glass icon on the right.

Examples Recent Google Drive **GitHub** Upload

Enter a GitHub URL or search by organization or user ☐ Include private repos

n-waves 🔍

Repository: [🔗](#) Branch: [🔗](#)

n-waves/multifit master

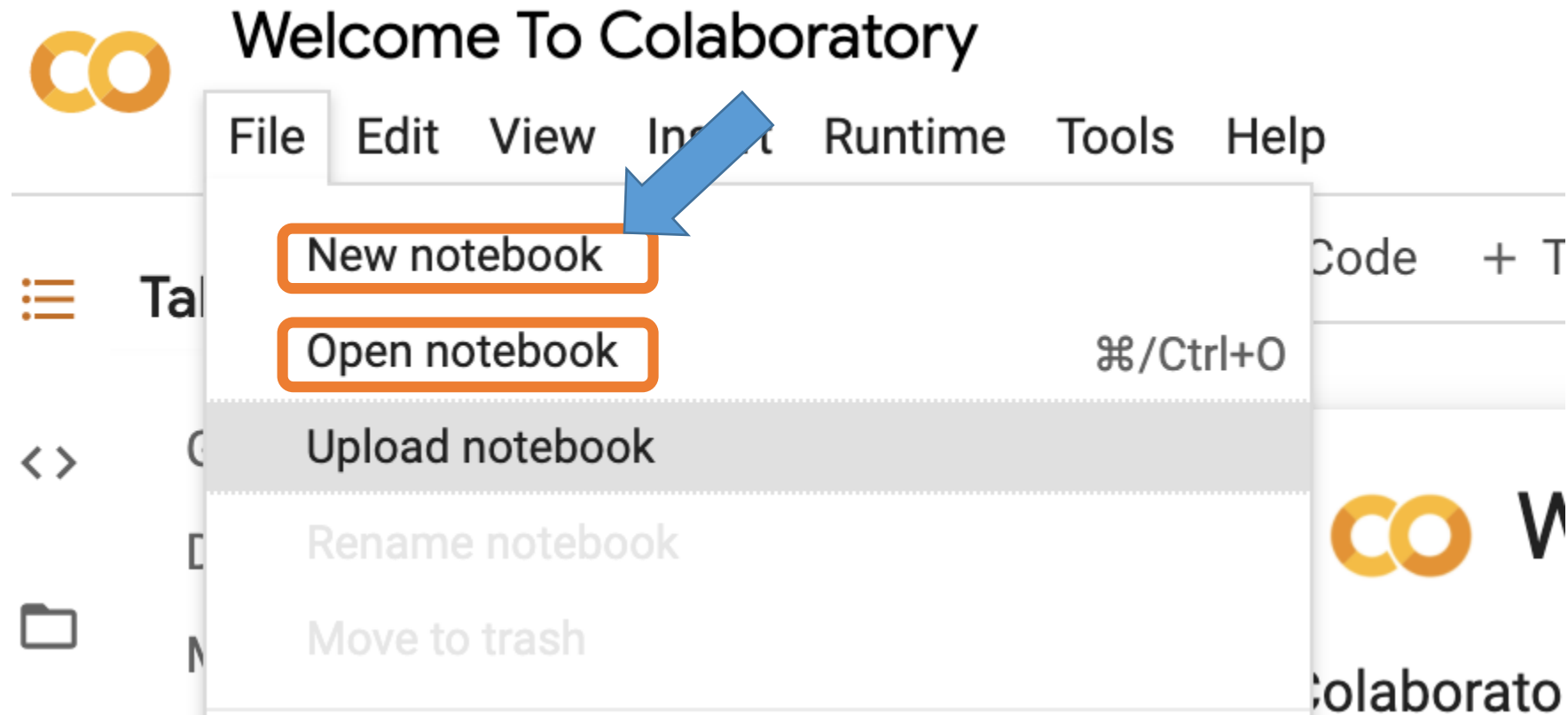
Path

notebooks/CLS-DE.ipynb 🔍 📄

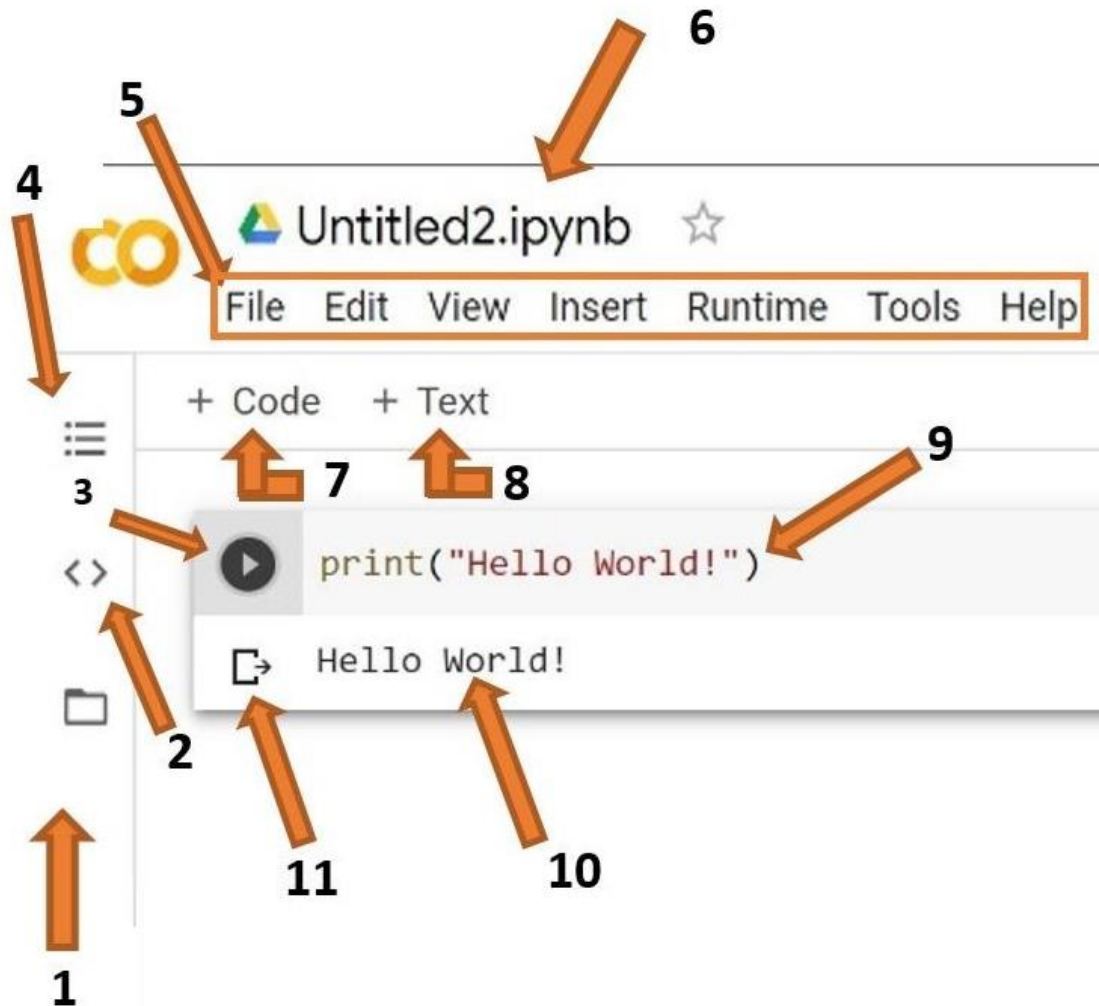
notebooks/CLS-JA.ipynb 🔍 📄

notebooks/MLDoc-JA-multifit_fp16.ipynb 🔍 📄

3. Starting a notebook

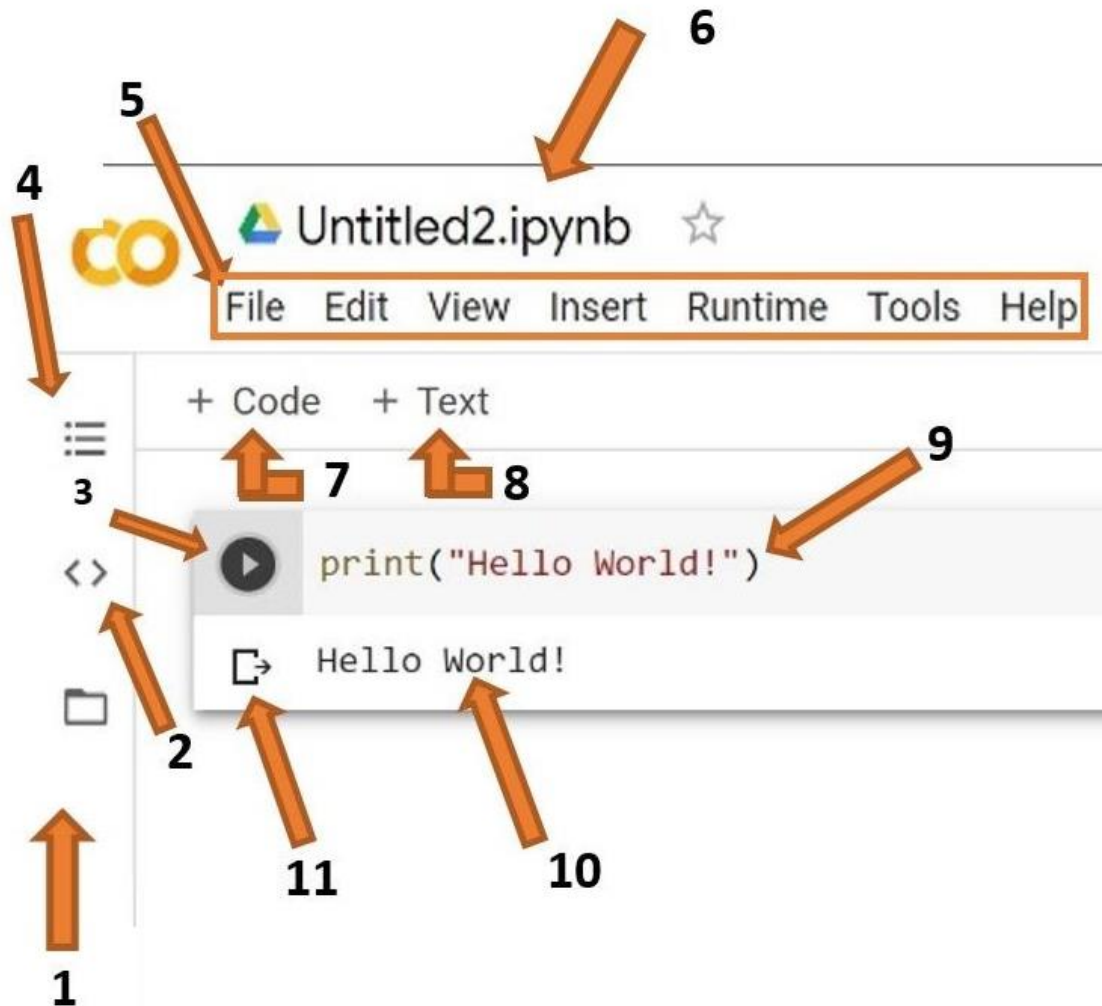


4. Toolbar



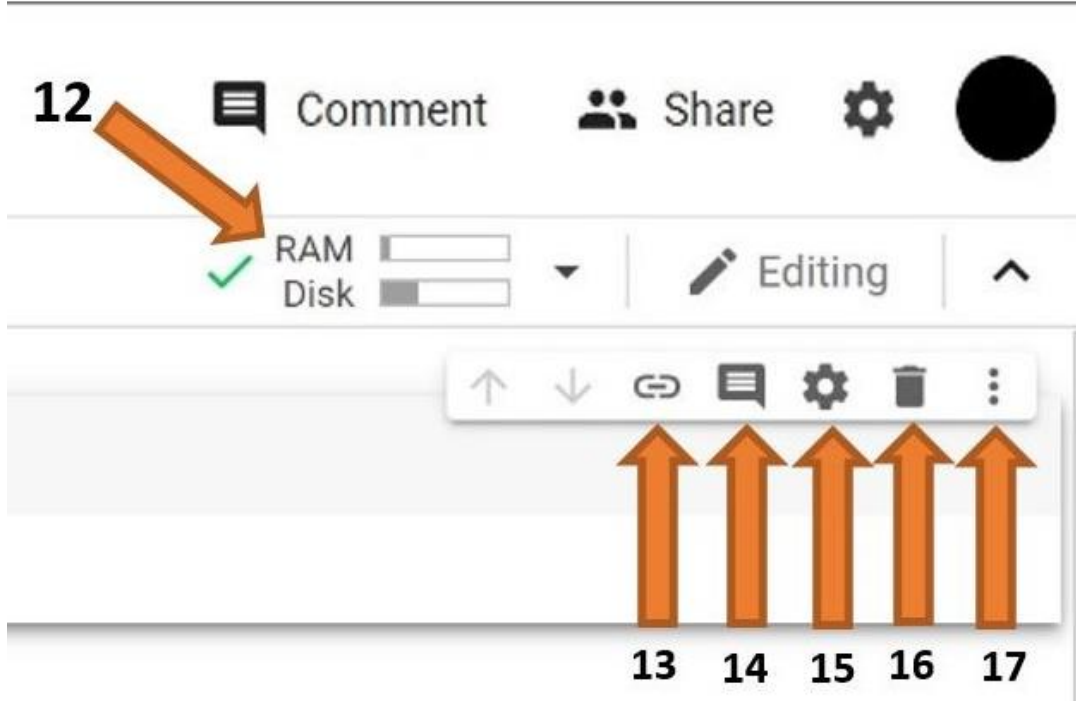
1. **Files:** Here you will be able to upload datasets and other files from both your computer and Google Drive
2. **Code Snippets:** Here you will be able to find prewritten snippets of code for different functionalities like adding new libraries or referencing one cell from another.
3. **Run Cell:** This is the run button. Clicking this will run any code that is inserted in the cell beside it. You can use the shortcut shift+enter to run the current cell and exit to a new one.
4. **Table of Contents:** Here you will be able to create and traverse different sections inside of your notebook. Sections allow you to organize your code and improve readability.

4. Toolbar



5. **Menu Bar:** Like in any other application, this menu bar can be used to manipulate the entire file or add new files.
6. **File Name:** This is the name of your file. You can click on it to change the name. Do not edit the extension (.ipynb) while editing the file name as this might make your file unopenable.
7. **Insert Code Cell:** This button will add a code cell below the cell you currently have selected.
8. **Insert Text Cell:** This button will add a text cell below the cell you currently have selected.
9. **Cell:** This is the cell. This is where you can write your code or add text depending on the type of cell it is.
10. **Output:** This is the output of your code, including any errors, will be shown.
11. **Clear Output:** This button will remove the output

4. Toolbar



- 12. Ram and Disk:** All of the code you write will run on Google's computer, and you will only see the output. This means that even if you have a slow computer, running big chunks of code will not be an issue. Google only allots a certain amount of Ram and Disk space for each *user*, so be mindful of that as you work on larger projects.
- 13. Link to Cell:** This button will create a URL that will link to the cell you have selected.
- 14. Comment:** This button will allow you to create a comment on the selected cell. Note that this will be a comment on (about) the cell and not a comment in the cell.
- 15. Settings:** This button will allow you to change the Theme of the notebook, font type, and size, indentation width, etc.
- 16. Delete Cell:** This button will delete the selected cell.
- 17. More Options:** Contains options to cut and copy a cell as well as the option to add form and hide code

Running blocks

- By pressing the Run button
- Shift + Enter – runs block
- Alt + Enter – creates a new block

Let us start

If you like to follow along, you can open your own notebook. But please try to keep up with my presentation, as you still have time for exercises after the teaching.

Agenda

- Variables
- Types
- Arithmetic operators
- Boolean logic
- Strings
- Printing
- Exercises

Python as a calculator

- Let us calculate the distance between Edinburgh and London in km

```
403 * 1.60934
```

```
648.56402
```

Variables

- Great calculator but how can we make it store values?
- Do this by defining variables
- Can later be called by the variable name
- Variable names are case sensitive and unique

```
distanceToLondonMiles = 403  
mileToKm = 1.60934  
distanceToLondonKm = distanceToLondonMiles * mileToKm  
distanceToLondonKm
```

648.56402

We can now reuse the variable mileToKm in the next block without having to define it again!

```
marathonDistanceMiles = 26.219  
marathonDistanceKm = marathonDistanceMiles * mileToKm  
print(marathonDistanceKm)
```

42.19528546

Types

Variables actually have a type, which defines the way it is stored.

The basic types are:

Type	Declaration	Example	Usage
Integer	int	<code>x = 124</code>	Numbers without decimal point
Float	float	<code>x = 124.56</code>	Numbers with decimal point
String	str	<code>x = "Hello world"</code>	Used for text
Boolean	bool	<code>x = True</code> or <code>x = False</code>	Used for conditional statements
NoneType	None	<code>x = None</code>	Whenever you want an empty variable

Why should we care?

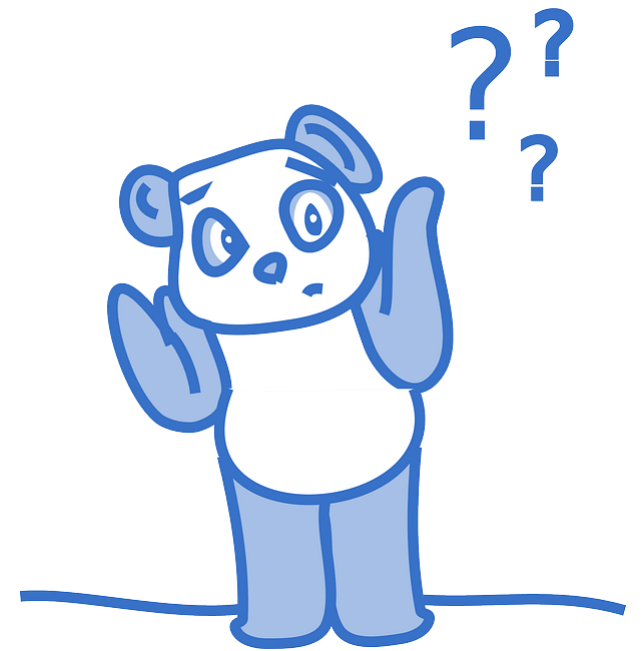


Image by [Cler-Free-Vector-Images on Pixabay](#)

```
In [4]: x = 10      # This is an integer
        y = "20"    # This is a string
        x + y
```

```
-----
-----
TypeError                                 Traceback (most recent call l
ast)
<ipython-input-4-f1463b8b4c2e> in <module>()
      1 x = 10      # This is an integer
      2 y = "20"    # This is a string
----> 3 x + y

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Important lesson to remember!

We can't do arithmetic operations on variables of different types. Therefore make sure that you are always aware of your variables types!

You can find the type of a variable using **type()**. For example type **type(x)**.

Casting types

Luckily Python offers us a way of converting variables to different types!

Casting – the operation of converting a variable to a different type

```
x = 10      # This is an integer  
y = "20"    # This is a string  
x + int(y)
```

30

Similar methods exist for other data types: **int()**, **float()**, **str()**

Quick quiz

```
x = "10"  
y = "20"  
x + y
```

What will be the result?

```
'1020'
```

Arithmetic operations

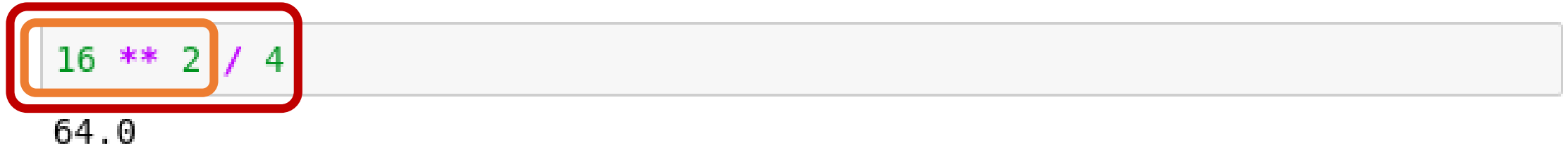
Similar to actual Mathematics.

Order of precedence is the same as in Mathematics.

We can also use parenthesis ()

Symbol	Task Performed	Example	Result
+	Addition	4 + 3	7
-	Subtraction	4 - 3	1
/	Division	7 / 2	3.5
%	Mod	7 % 2	1
*	Multiplication	4 * 3	12
//	Floor division	7 // 2	3
**	Power of	7 ** 2	49

Order precedence example



Quick quiz

4 + 3 ** 2

13

vs

(4 + 3) ** 2

49

Comparison operators

- I.e. comparison operators
- Return Boolean values (i.e. True or False)
- Used extensively for conditional statements

Operator	Output
$x == y$	True if x and y have the same value
$x != y$	True if x and y don't have the same value
$x < y$	True if x is less than y
$x > y$	True if x is more than y
$x <= y$	True if x is less than or equal to y
$x >= y$	True if x is more than or equal to y

Comparison examples

```
x = 5      # assign 5 to the variable x  
x == 5     # check if value of x is 5
```

True

Note that `==` is not the same as `=`

```
x > 7
```

False

Logical operators

- Allows us to extend the conditional logic
- Will become essential later on

Operation	Result
x or y	True if at least one is True
x and y	True only if both are True
not x	True only if x is False

a	not a	a	b	a and b	a or b
False	True	False	False	False	False
True	False	False	True	False	True
		True	False	False	True
		True	True	True	True

Truth-table definitions of bool operations

Combining both

```
x = 14  
# check if x is within the range 10..20
```

True and **True**

True

Another example

```
x = 14  
y = 42  
not (  )  
False
```

That wasn't very easy to read was it?
Is there a way we can make it more readable?

```
x = 14
y = 42

xDivisible = ( x % 2 ) == 0 # check if x is a multiple of 2
yDivisible = ( y % 3 ) == 0 # check if y is a multiple of 3

not (xDivisible and yDivisible)
```

False

Strings

- Powerful and flexible in Python
- Can be added
- Can be multiplied
- Can be multiple lines

Strings

```
x = "Python"  
y = "rocks"  
x + " " + y
```

'Python rocks'

```
x = "This can be"  
y = "repeated "  
x + " " + y * 3
```

'This can be repeated repeated repeated '

Strings

```
x = "Edinburgh"  
x = x.upper()  
  
y = "University Of "  
y = y.lower()  
  
y + x  
  
'university of EDINBURGH'
```

These are called methods and add extra functionality to the String.
If you want to see more methods that can be applied to a string simply type in **dir('str')**

Mixing up strings and numbers

Often we would need to mix up numbers and strings.
It is best to keep numbers as numbers (i.e. int or float)
and cast them to strings whenever we need them as a string.

```
x = 6
x = ( x * 5345 ) // 63
"The answer to Life, the Universe and Everything is " + str(x)
'The answer to Life, the Universe and Everything is 42'
```

Multiline strings

```
x = """To include  
multiple lines  
you have to do this"""  
y = "or you can also\ninclude the special\ncharacter '\\n' between lines"  
print(x)  
print(y)
```

To include
multiple lines
you have to do this
or you can also
include the special
character `\n` between lines

Printing

- When writing scripts, your outcomes aren't printed on the terminal.
- Thus, you must print them yourself with the print() function.
- Beware to not mix up the different type of variables!

```
print("Python is powerful!")
```

Python is powerful!

```
x = "Python is powerful"  
y = " and versatile!"  
print(x + y)
```

Python is powerful and versatile!

Quick quiz

Do you see anything wrong with this block?

```
str1 = "which means it has even more than"  
str2 = 76  
str3 = "quirks"  
print(str1 + str2 + str3)
```

```
-----  
-----  
TypeError                                 Traceback (most recent call l  
ast)  
<ipython-input-2-3be15a6244a4> in <module>()  
      2 str2 = 76  
      3 str3 = " quirks"  
----> 4 print(str1 + str2 + str3)  
  
TypeError: must be str, not int
```

Another more generic way to fix it

```
str1 = "It has"  
str2 = 76  
str3 = "methods!"  
print(str1, str2, str3)
```

It has 76 methods!

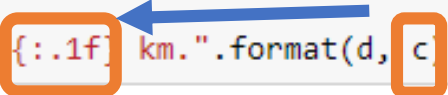
If we comma separate statements in a print function we can have different variables printing!

Placeholders

- A way to interleave numbers is

```
pi = 3.14159 # Pi
d = 12756 # Diameter of eath at equator (in km)
c = pi*d # Circumference of equator

#Print using +, and casting
print("Earth's diameter at equator: " + str(d) + "km. Equator's circumference:" + str(c) + "km.")
#Print using several arguments
print("Earth's diameter at equator:", d, "km. Equator's circumference:", c, "km.")
#Print using .format
print("Earth's diameter at equator: {:.1f} km. Equator's circumference: {:.1f} km.".format(d, c))
```



```
Earth's diameter at equator: 12756km. Equator's circumference:40074.12204km.
Earth's diameter at equator: 12756 km. Equator's circumference: 40074.12204 km.
Earth's diameter at equator: 12756.0 km. Equator's circumference: 40074.1 km.
```

- Elegant and easy
- more in your notes

Commenting

- Useful when your code needs further explanation. Either for your future self and anybody else.
- Useful when you want to remove the code from execution but not permanently
- Comments in Python are done with #
 - `print(totalCost)` is ambiguous and we can't exactly be sure what `totalCost` is.
 - `print(totalCost) # Prints the total cost for renovating the Main Library` is more informative

Failure is progress!

Ask me anything. Ask among yourselves as well.

GOT A NEW ERROR



PROGRESS

memegenerator.net