

# I Introduction to Python



Data Science  
Academy

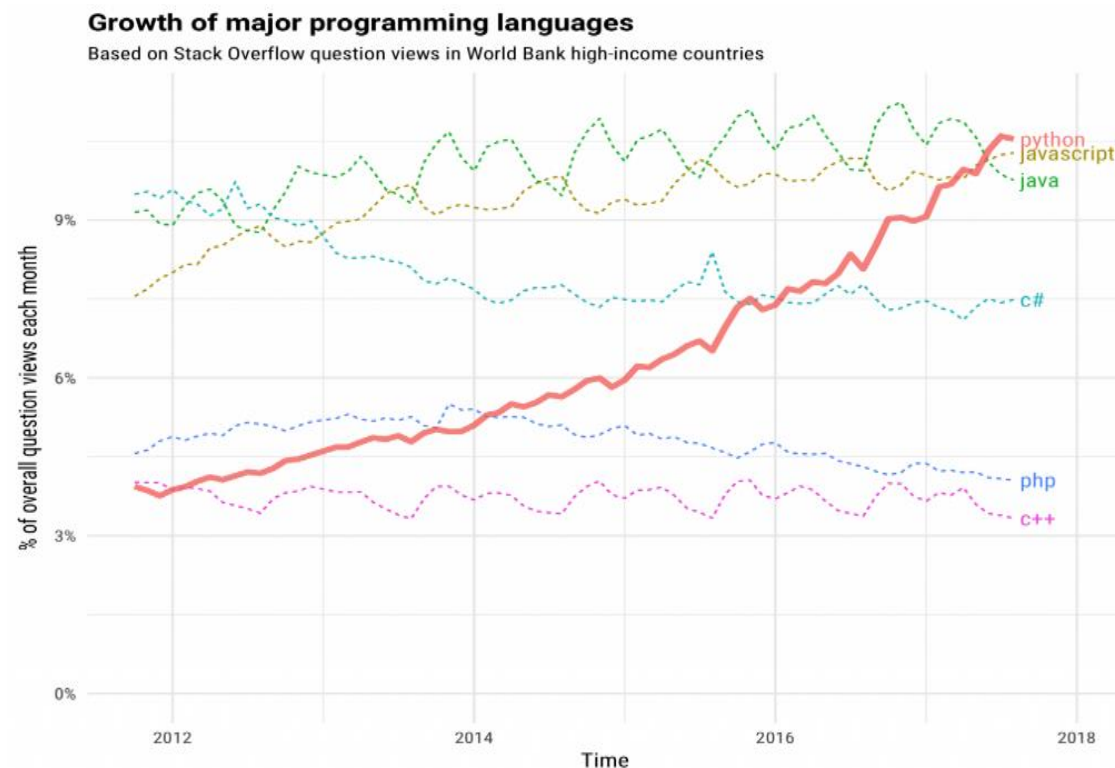
# Python

- Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes **code readability** with its notable use of significant whitespace
- Python is an interpreted, high-level, general-purpose programming language.
  - **Interpreted** – Instructions are executed directly without being compiled into machine-language instructions. Compiled languages unlike interpreted languages, are faster and give the developer more control over memory management and hardware resources.
  - **High-level** – allowing us to perform complex tasks easily and efficiently



# Why Python for Data Science

- Python competes with many languages in the Data Science world, most notably R and to a much lesser degree MATLAB, JAVA and C++.



# Companies using python

The popular YouTube video sharing system is largely written in Python



Google makes extensive use of Python in its web search system



Dropbox storage service codes both its server and client software primarily in Python



The Raspberry Pi single-board computer promotes Python as its educational language



COMPANIES USING PYTHON



BitTorrent peer-to-peer file sharing system began its life as a Python Program



NASA uses Python for specific Programming Task



The NSA uses Python for cryptography and intelligence analysis

**NETFLIX**

Netflix and Yelp have both documented the role of Python in their software infrastructures

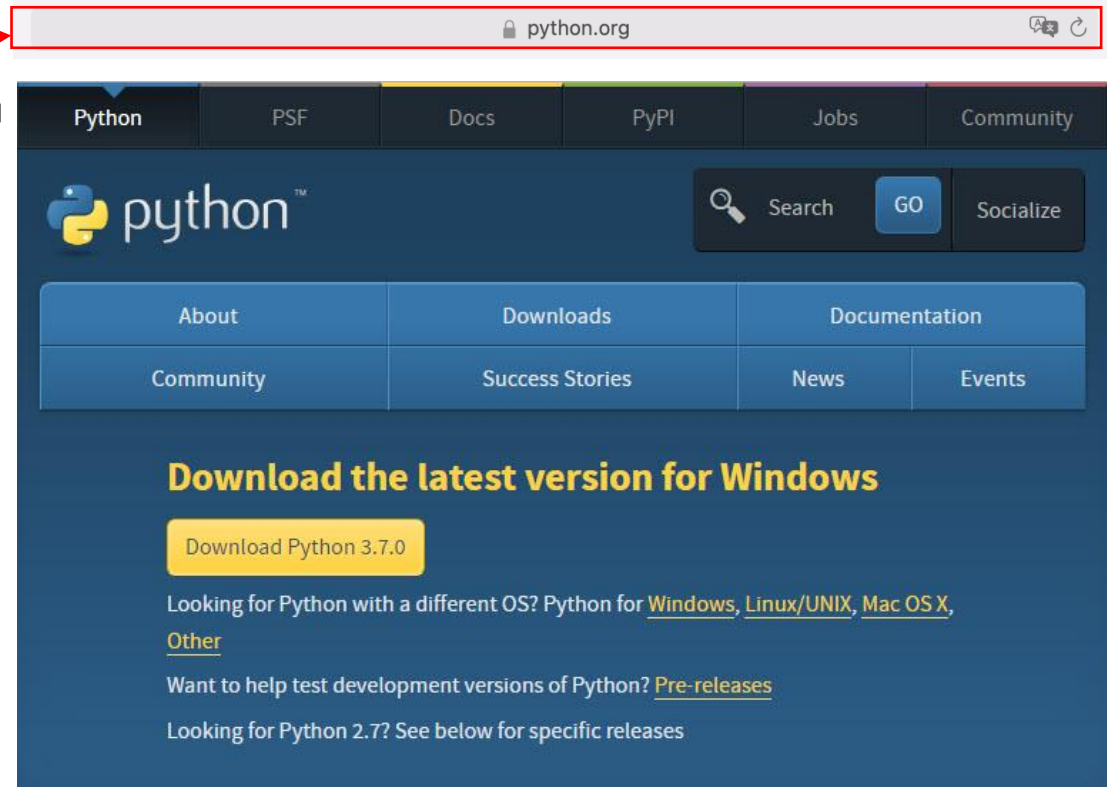
# Why does Python Lead the Pack?

- It is the only general-purpose programming language that comes with a solid ecosystem of scientific computing libraries.
- Supports a number of popular Machine Learning, Statistical and Numerical Packages (Pandas, Numpy, Scikit-learn, TensorFlow, Cython)
- Supports easy to use iPython Notebooks, especially handy for view Data Science work.
- Quite easy to get started
- As a general purpose language it allows more flexibility such as building web servers, APIs and a plethora of other useful programming libraries.

# Install Python

Check the given link and click [“Downloads”](#) section

- Choose operating system of your computer
- Download Python 3.7 version



# What is Anaconda?

- Free and open-source distribution of Python and R
- Predominantly used for Data Science, Machine Learning and large-scale data processing
- Over 12 million users
- 1400 packages



# System Requirements

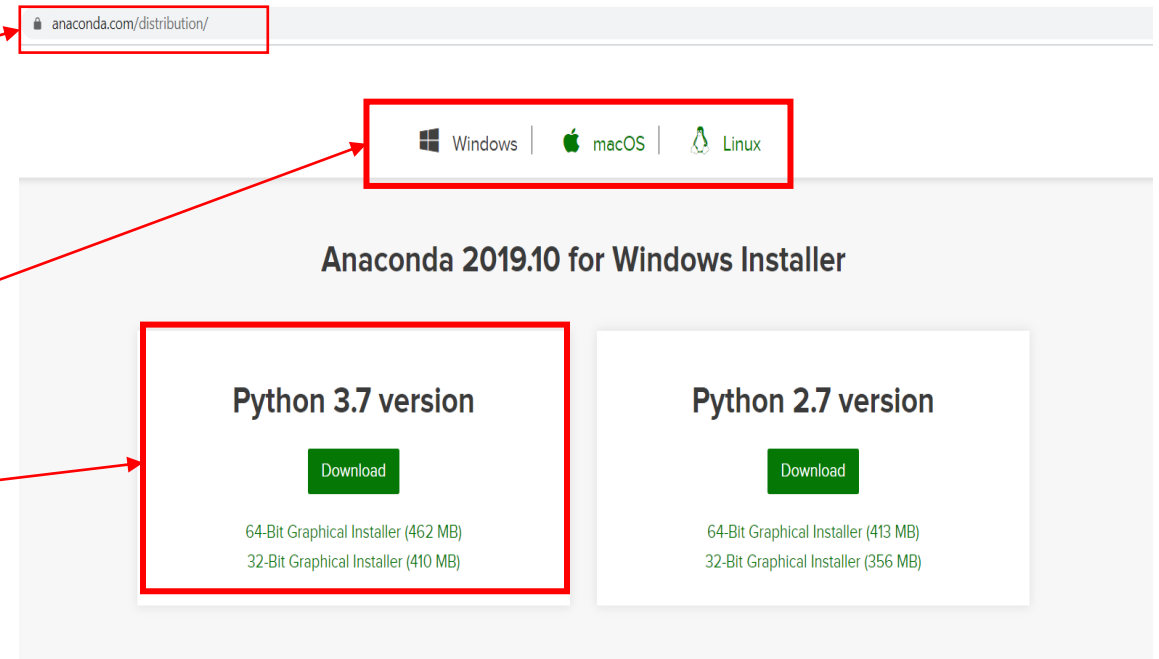
- Operating system: Windows 7 or newer, 64-bit macOS 10.10+, or Linux, including Ubuntu, RedHat, CentOS 6+, and others.
- Older versions of Anaconda are available in archive
- System architecture: Windows- 64-bit x86, 32-bit x86; MacOS- 64-bit x86; Linux- 64-bit x86, 64-bit Power8/Power9.
- Minimum 5 GB disk space to download and install.



# Install Anaconda

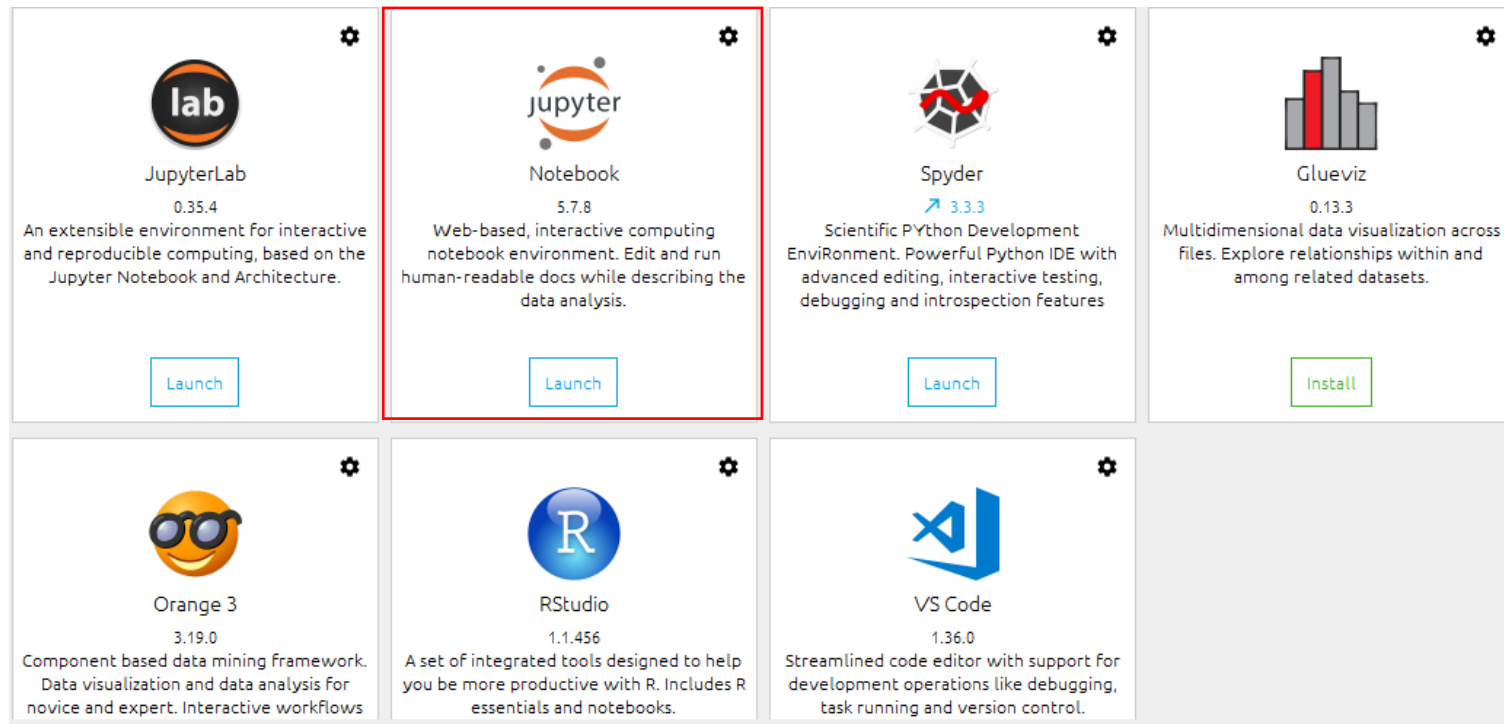
Check the given link and download [“Anaconda”](#)

- Choose operating system of your computer
- Download Python 3.7 version









# Install Jupyter Notebook

- Run Anaconda
- Open “jupyter Notebook”

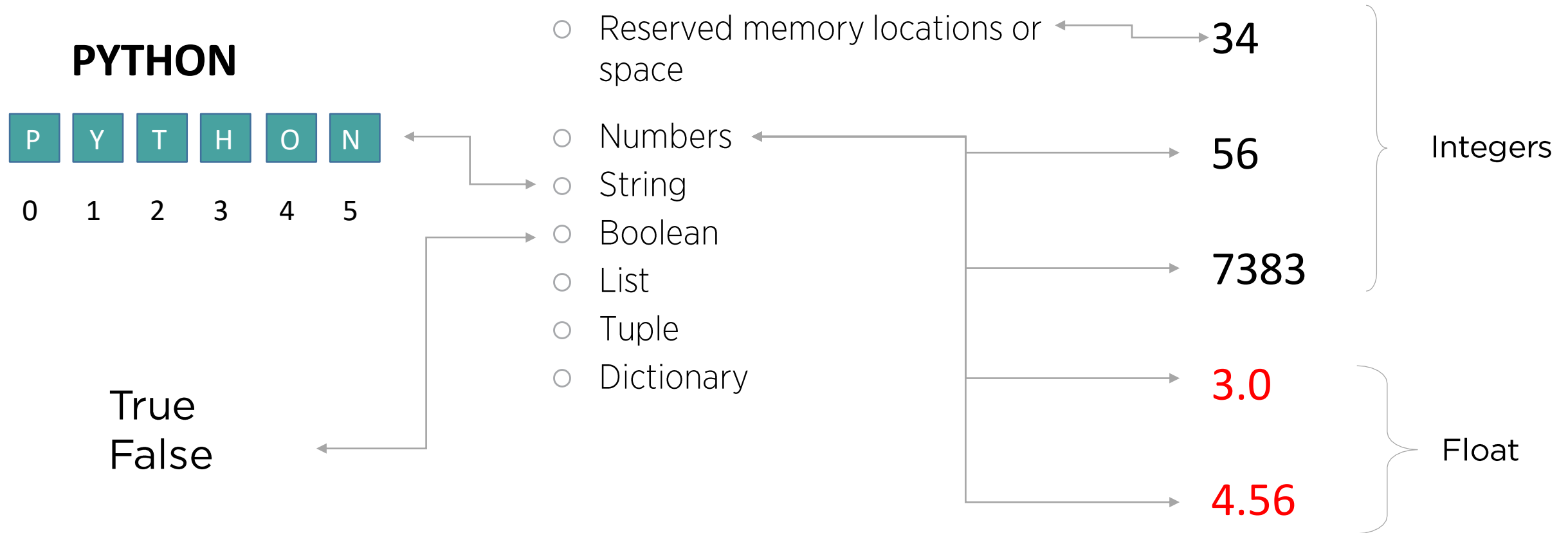


# Or try Jupyter online

- Go to <https://jupyter.org/try>
- Select one of the examples on the page

<b>Try Classic Notebook</b>  A tutorial introducing basic features of Jupyter notebooks and the IPython kernel using the classic Jupyter Notebook interface.	<b>Try JupyterLab</b>  JupyterLab is the new interface for Jupyter notebooks and is ready for general use. Give it a try!	<b>Try Jupyter with Julia</b>  A basic example of using Jupyter with Julia.
<b>Try Jupyter with R</b>  A basic example of using Jupyter with R.	<b>Try Jupyter with C++</b>  A basic example of using Jupyter with C++	<b>Try Jupyter with Scheme</b>  Explore the Calysto Scheme programming language, featuring integration with Python

# Data types in Python - Variables



II

# Simple Values and **Data Structures**

# Integer

In Python 3, there is effectively no limit to how long an integer value can be. Of course, it is constrained by the amount of memory your system has, as are all things, but beyond that an integer can be as long as you need it to be.

**123**

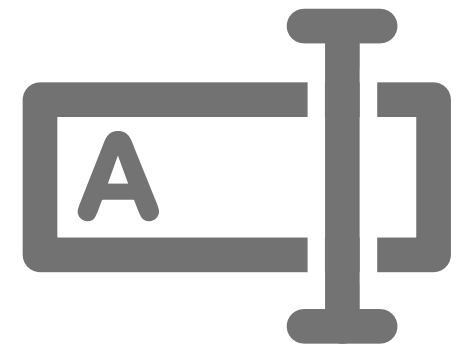
# Float

The float type in Python designates a floating-point number. Float values are specified with a decimal point. Optionally, the character `e` or `E` followed by a positive or negative integer may be appended to specify scientific notation

# 7.3

# String

- Strings are sequences of character data. The string type in Python is called `str`.
- A string in Python can contain as many characters as you wish. The only limit is your machine's memory resources. A string can also be empty.
- Strings are immutable that means once defined they cannot be changed





# Boolean

- Boolean values are the two constant objects *False* and *True*.
- In numeric contexts (for example, when used as the argument to an arithmetic operator), they behave like the integers 0 and 1, respectively.
- The `bool()` function allows you to evaluate any value, and give you *True* or *False* in return.



# List

- Sequence of elements
- Similar to array in other programming languages
- Elements are indexed

```
List1 = [ "Ali", "Namiq", "Leyla", "Gunay"]
```

0          1          2          3

# Tuple

- Sequence of elements
- Similar to array in other programming languages
- Tuples are immutable
- Faster to process
- Elements are indexed

```
Tuple1 = ("Ali", "Namiq", "Leyla", "Gunay")
```

0      1      2      3

# Dictionary

- Collection of key-value pairs
- Values can be accessed using the Key
- Used for JSON format conversion

Address

Key	Value
Street	Ashiq Ali 2
City	Baku
Region	Absheron
Country	Azerbaijan

Address = {'Street': 'Ashiq Ali 2', 'City': 'Baku', 'Region': 'Absheron', 'Country': 'Azerbaijan'}

Address['Street'] = 'Ashiq Ali 2'

Address['Region'] = 'Absheron'

III

# Functions and Conditionals



Data Science  
Academy

# Add Comments

- **Comments** are sentences not executed by the computer; it doesn't read them as instructions. The trick is to put a *hash sign* at the beginning of each line you would like to insert as a comment.
- If you would like to leave a comment on two lines, don't forget to place the hash sign at the beginning of each line.
- Or another thing you can do is use multiline strings by wrapping your comment inside a set of triple quotes, that's called documentation string

```
In [1]: #This is just a comment and not code!  
        print (7,2)
```

7 2

```
In [2]: #Comment 1  
        #Comment 2  
        print (1)
```

1

```
"""  
If I really hate pressing 'enter' and typing all those  
hash marks, I could just do this instead  
"""
```

# Arithmetic Operators

Operator	Description
==	Verifies the left and right side of an equality are equal
!=	Verifies the left and right side of an equality are not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
+	Sum (plus)
-	Subtraction (minus)
/	Division
*	Multiplication
**	Superpower

# Structure Code with Indentation

- The way you apply **indentation** in practice is important, as this will be the only way to communicate your ideas to the machine properly.
- *Def* and *Print* form two separate and, written in this way, clearly distinguishable **blocks of code** or **blocks of commands**.
- Everything that regards the function is written with one indentation to the inside. Once you decide to code something else, start on a new line with no indentation.

```
In [1]: def five(x):  
        x = 5  
        return x  
        print five(3)
```

block of code/command #1

block of code/command #2



# Defining a Function in Python

- Write **def** at the beginning of the line. Def is neither a command nor a function. It is a **keyword**. To indicate this, Jupyter will automatically change its font color to green.
- 1. Type **name of the function**.
- 2. Add a pair of **parentheses**.
- 3. Place **parameters** of the function if it requires you to have any. It is possible to have a function with zero parameters.
- 4. Put a **colon** after the name of the function.
- Since it is inconvenient to continue on the same line when the function becomes longer, it is much better to build the habit of laying the instructions on a new line, with an **indent** again.

```
In [1]: def simple():  
        print ( "My first function" )
```

def function\_name (parameters) :  
 ↔ function body

# Creating a Function with a Parameter

- **return** a value from the function. *plus\_ten(a)* to do a specific calculation and not just print something.
- Define a function, we specify in parentheses a **parameter**. In the *plus\_ten()* function, “a” is a parameter. When we call this function, it is correct to say we provide an **argument**, not a parameter. So we can say “call *plus\_ten()* with an argument of 2, call *plus\_ten()* with an argument of 5”.

```
In [3]: def plus_ten (a):  
        return a + 10
```

```
In [4]: plus_ten(2)
```

```
Out [4]: 12
```

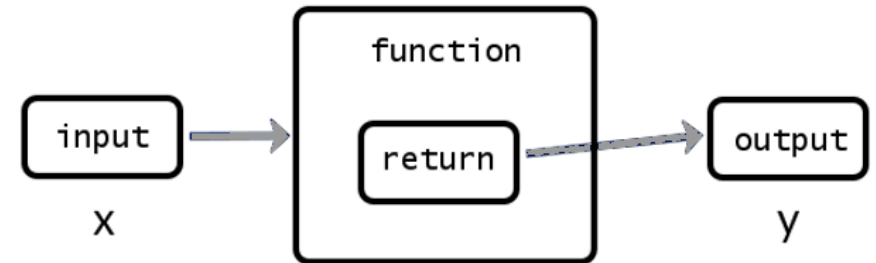
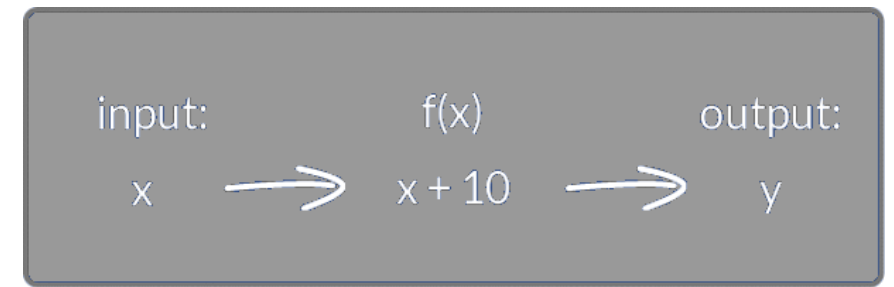
```
In [5]: plus_ten(5)
```

```
Out [4]: 15
```

**Def** function\_name (parameters) :  
But call  
plus\_ten (arguments) :

# Creating a Function with a Parameter

- **return** regards the value of  $y$ ; it just says to the machine “after the operations executed by the function  $f$ , return to me the value of  $y$ ”. “Return” plays a connection between the second and the third step of the process.
- A function can take an input of one or more variables and return a **single** output composed of one or more values.
- This is why “return” can be used only once in a function.



# Print vs Return

- “Print” takes a statement or, better, an object, and provides its printed representation in the output cell. It just makes a certain statement visible to the programmer. “Print” does not affect the calculation of the output.
- “Return” does not visualize the output. It specifies what a certain function is supposed to give back.

Print	VS.	Return
Does not affect the calculation of the output		Does not visualize the output  It specifies what a certain function is supposed to give back

# Indexing

- Is it possible to extract the letter “d”?
  - Yes, by using square brackets. Specify the position of the letter we would like to be extracted.

**Note:** Make sure you don't mistake brackets for parentheses or braces:

parentheses – () brackets – [] braces– {}

```
In [ ]: "Friday"[]
```

Name\_of\_variable[index\_of\_element]

# Indexing

*Note:* in Python, we count from 0, not from 1!

- 0,1,2,3,4, and so on.
- That's why I'll ask for the 4<sup>th</sup> letter, 'd', by writing 3 here.

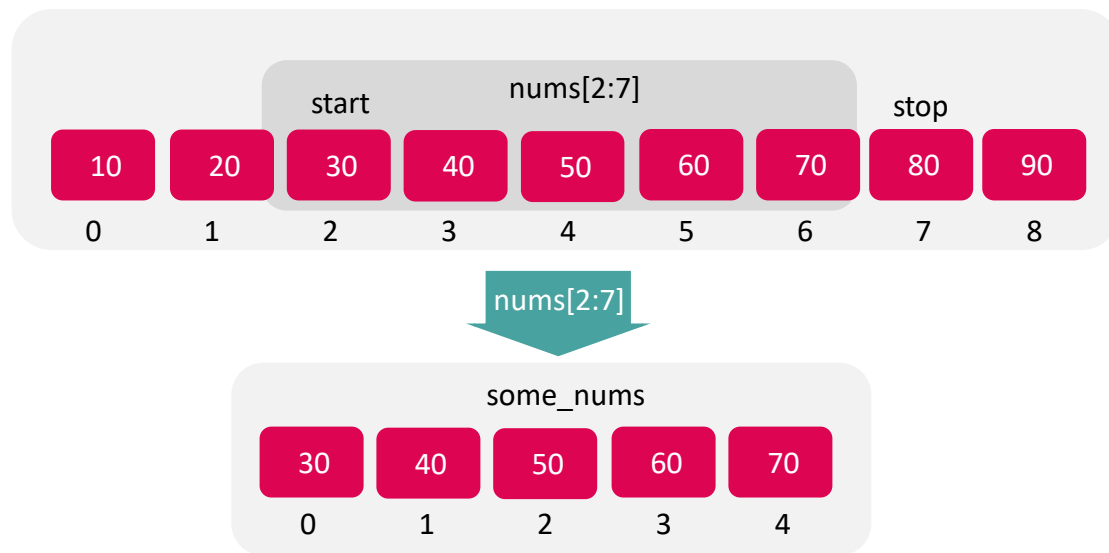
```
In [1]: "Friday"[3]
```

```
Out [1]: 'd'
```

F	R	I	D	A	Y
0	1	2	3	4	5

# Slicing

As it was shown, indexing allows you to access/change/delete only a single cell of a list. What if we want to get a sublist of the list? This is a snap when using slice:



```
In [1]: nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
```

```
In [2]: some_nums = nums [2:7]
```

```
In [3]: some_nums
```

```
Out [3]: [30, 40, 50, 60, 70]
```

# Logical Operators

- The **logical operators** in Python are the words “not”, “and”, and “or”.
- They compare a certain amount of statements and return Boolean values – “True” or “False” – hence their second name, **Boolean operators**.

Operator	Description
<b>“And”</b>	Checks whether the two statements around it are “True” <i>Example: “True and False” leads to True</i>
<b>“Or”</b>	Checks whether at least one of the two statements is “True” <i>Example: “False or True” leads to True</i>
<b>“Not”</b>	Leads to the opposite of the given statement <i>Example: “not True” leads to False</i>



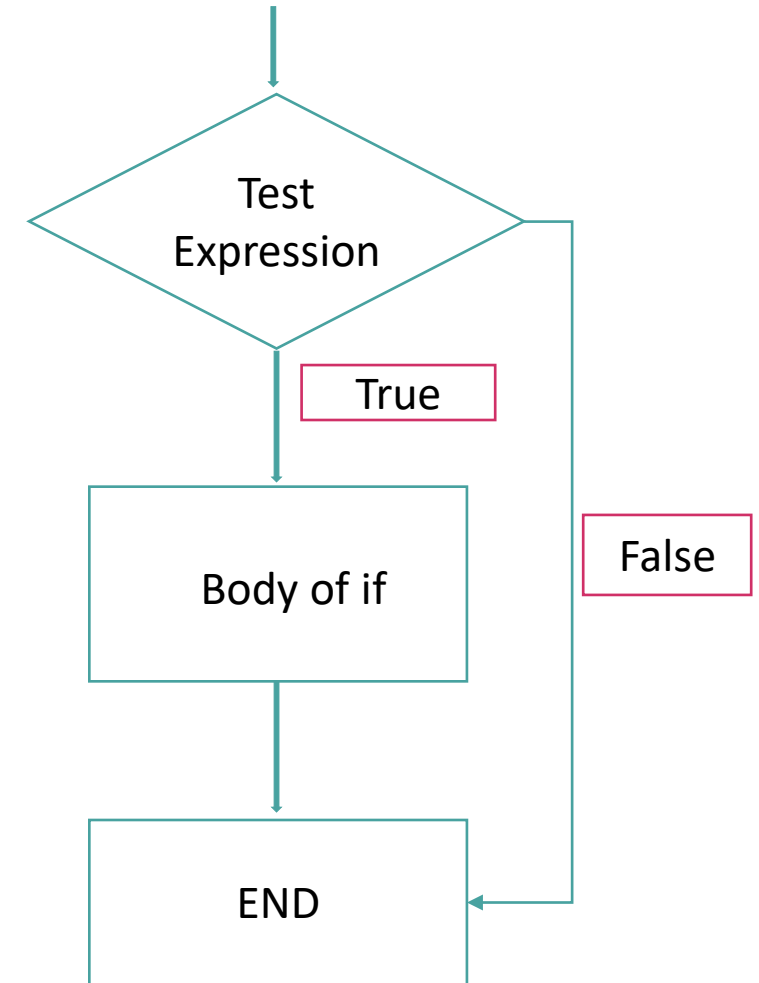
# Precedence of Operators

- The operator precedence in Python is listed in the following table. It is in descending order (upper group has higher precedence than the lower ones).

Operator	Description
()	Parentheses
**	Exponent
+x, -x, ~x	Unary plus, Unary minus, Bitwise NOT
*, /, //, %	Multiplication, Division, Floor Division, Modulus
+, -	Addition, Subtraction
<<, >>	Bitwise shift operators
&	Bitwise AND
^	Bitwise XOR
	Bitwise OR
==, !=, >, >=, <, <=, is, is not, in, not in	Comparisons, Identity, Membership operators
not	Logical NOT
and	Logical AND
or	Logical OR

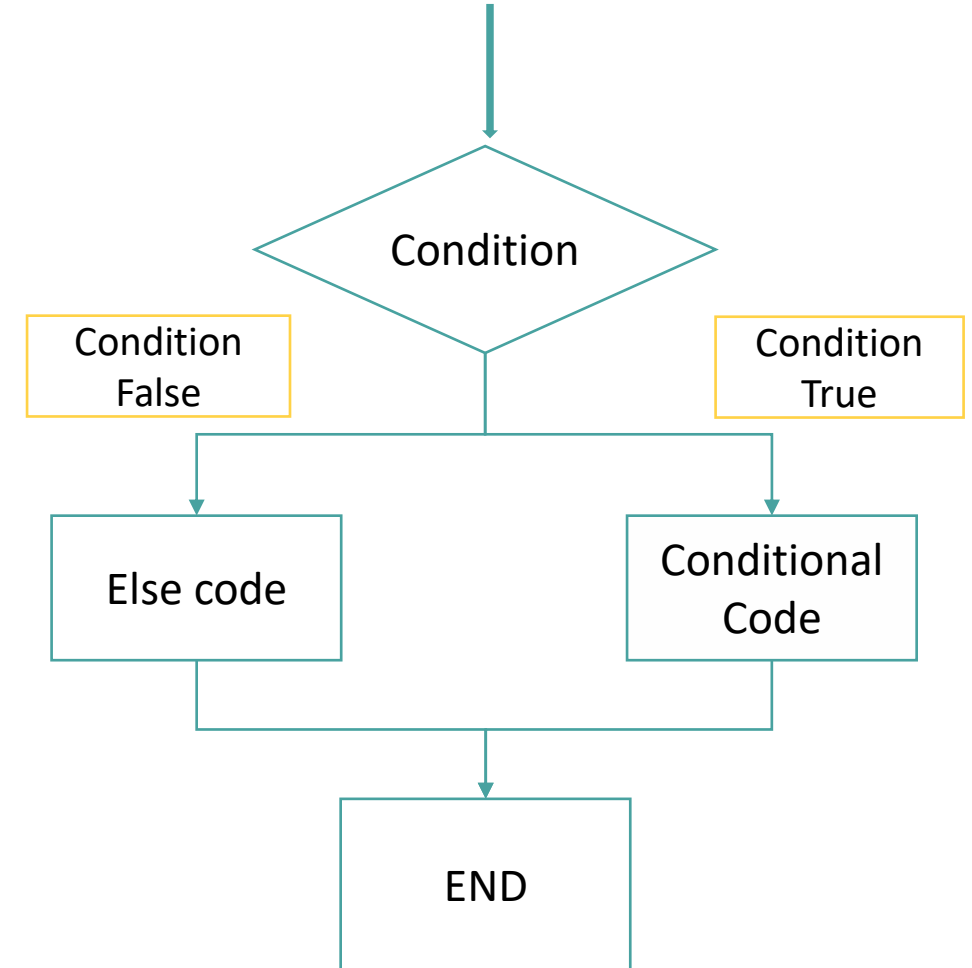
# What is if...else statements?

- Decision making is required when we want to execute a code only if a certain condition is satisfied.
- The if...elif...else statement is used in Python for decision making.



# Add an ELSE statement

- If the condition is false, result is *else code*.
- Whether the initial condition is satisfied, we will get to the end point, so the computer has concluded the entire operation and is ready to execute a new one.



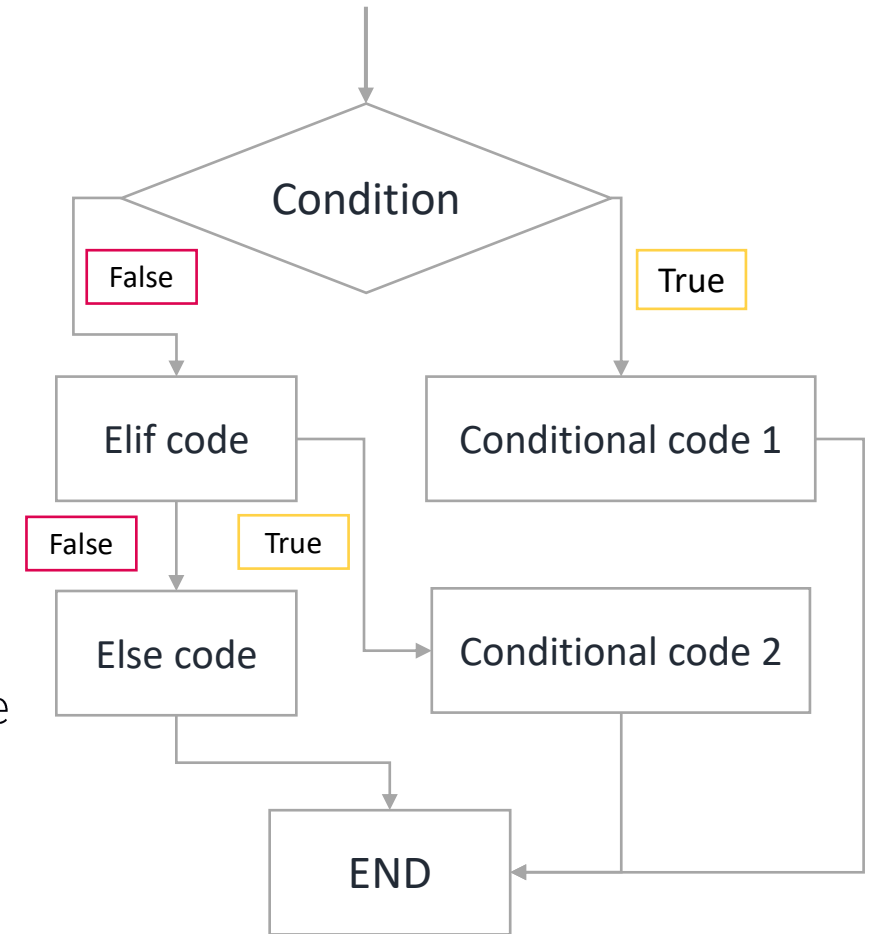
# Else if, for Brief - ELIF

- If y is not greater than 5, the computer will think: “else if y is less than 5”, written “**elif** y is less than 5”, then I will print out “Less”.

```
In [1]: def compare_to_five (y) :  
        if y > 5:  
            return "Greater"  
        elif y < 5:  
            return "Less"  
        else:  
            return "Equal"
```

# Else if, for Brief - ELIF

- Computer always reads your commands from top to bottom. This is something like the flow of the logical thought of the computer, the way the computer thinks – step by step, executing the steps in a rigid order.
- When it works with a conditional statement, the computer's task will be to execute a specific command once a certain condition has been satisfied. It will read commands from the if- statement at the top, through the elif-statements in the middle, to the else- statement at the end. The first moment the machine finds a satisfied condition, it will print the respective output and will execute no other part of the code from this conditional.



# Thank You