Intro to Computing

Python 101 - Week 1

Mehmet Arif Demirtaş
marifdemirtas.github.io

About the Course

Class: Monday 18.00 @ EEB 5102

Instructor(s):

- Mehmet Arif Demirtaş

https://marifdemirtas.github.io/

- Tolga Kılınçkaya

https://www.linkedin.com/in/tolga-k%C4%B1l%C4%B1n%C3%A7kaya-42a49a1b1/

Topics:

- October 10: **Introduction**
- October 17: **Data Types + Control Flow**
- October 24: **Collections**
- October 31: **Functions**
- November 7: **TERM BREAK**
- November 14: **Object Oriented Programming**
- November 21: **Wrap-up**

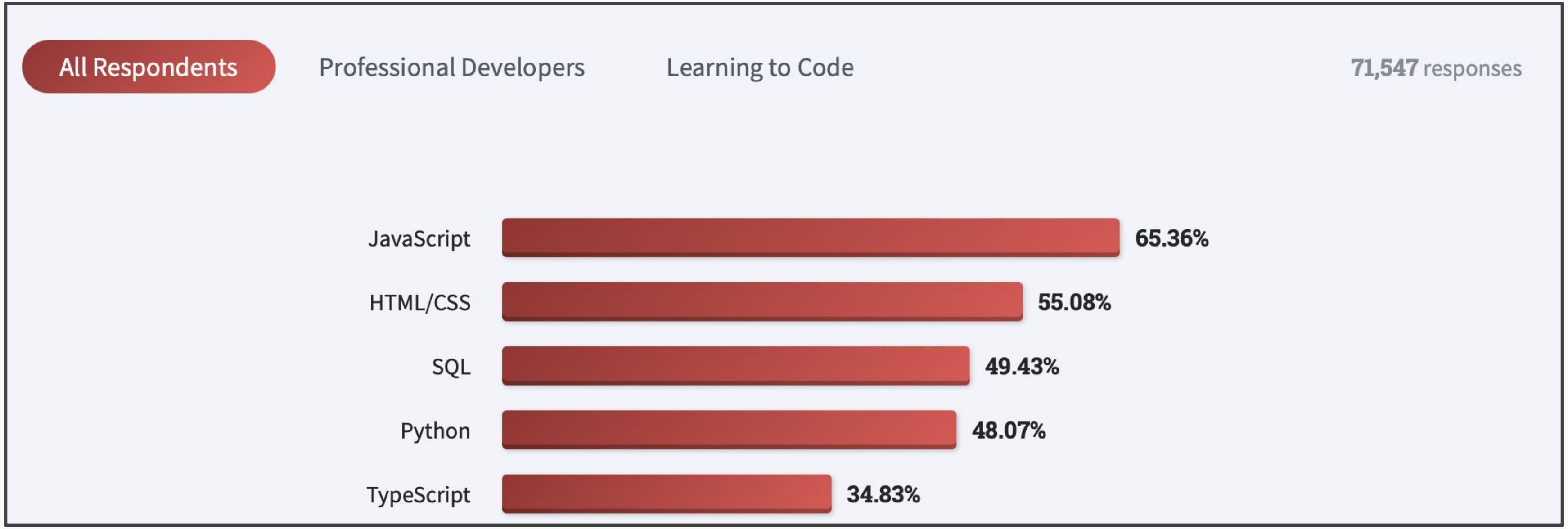
Book: Guttag, John. Introduction to Computation and Programming Using Python: With Application to Understanding Data Second Edition. MIT Press, 2016. ISBN: 9780262529624 *(First eight chapters)*

Today's Plan

- Why Python?
- What is a computer?
- What is a computer program?
- Fundamental set of operations in a computer program
- Additional operations in computer programs
- Coding in Python

Motivation - Why Python?

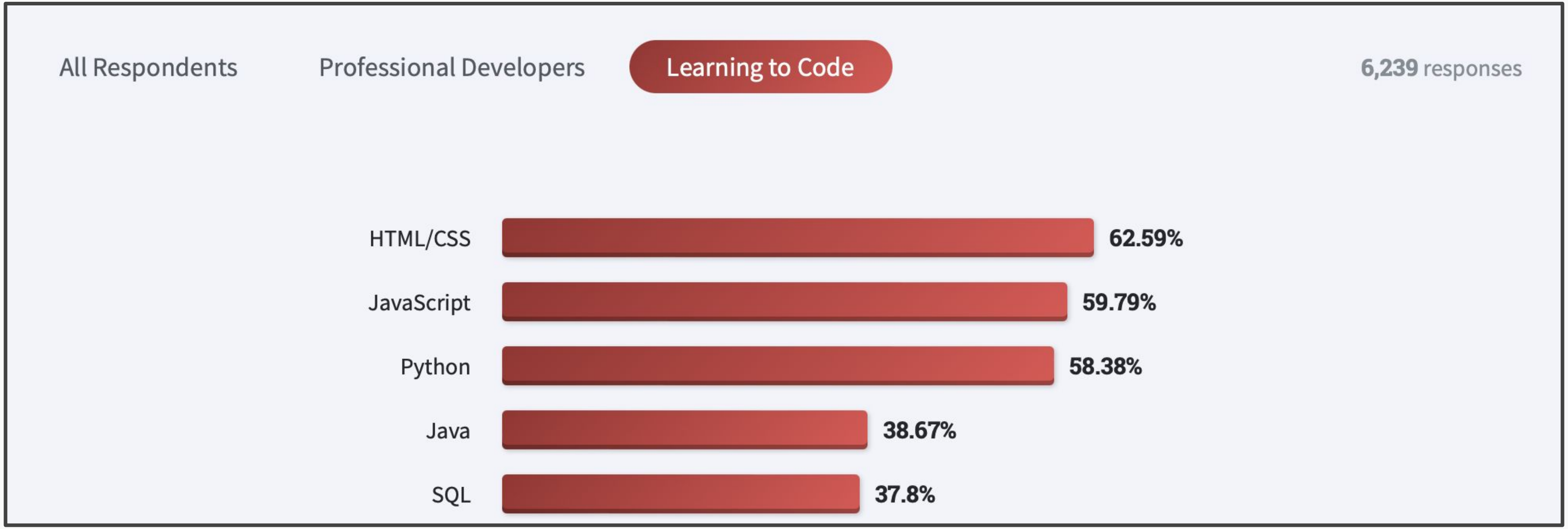
Which programming, scripting, and markup languages have you done extensive development work in over the past year, and which do you want to work in over the next year?



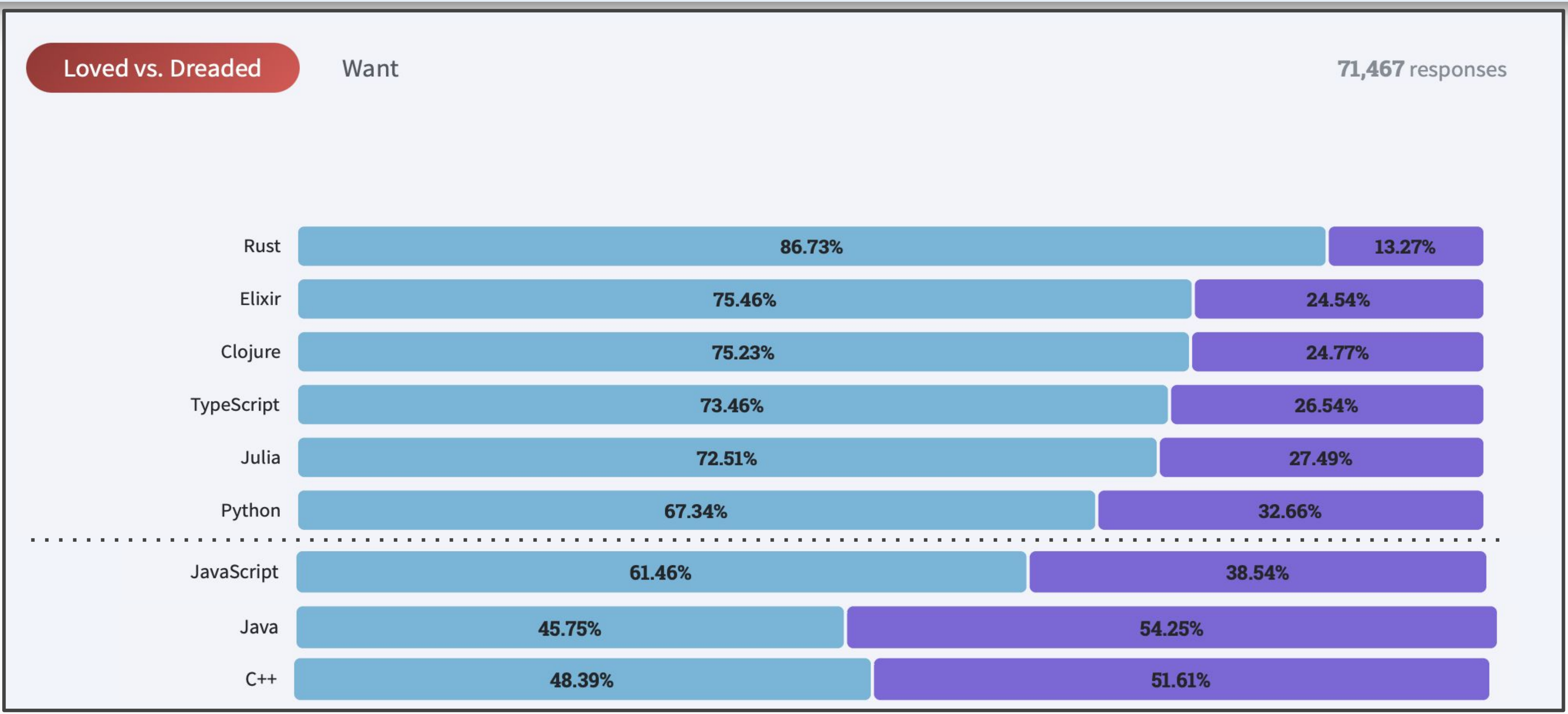
https://survey.stackoverflow.co/2022/

Motivation - Why Python?

Which programming, scripting, and markup languages have you done extensive development work in over the past year, and which do you want to work in over the next year?

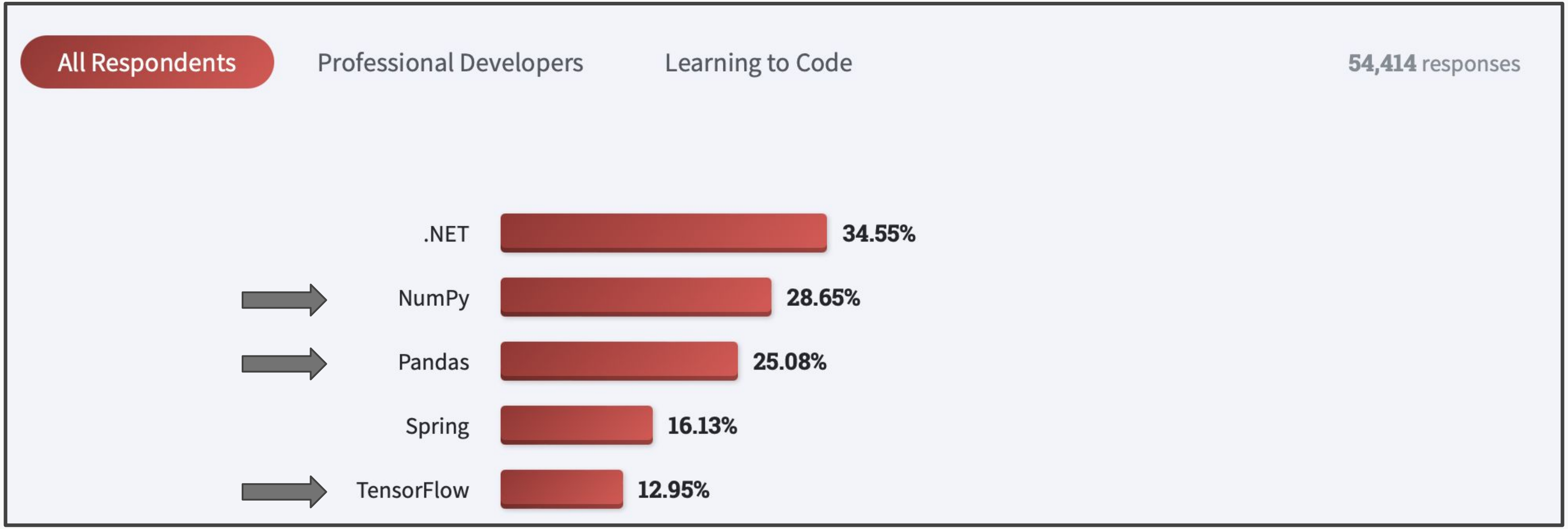


Motivation - Why Python?

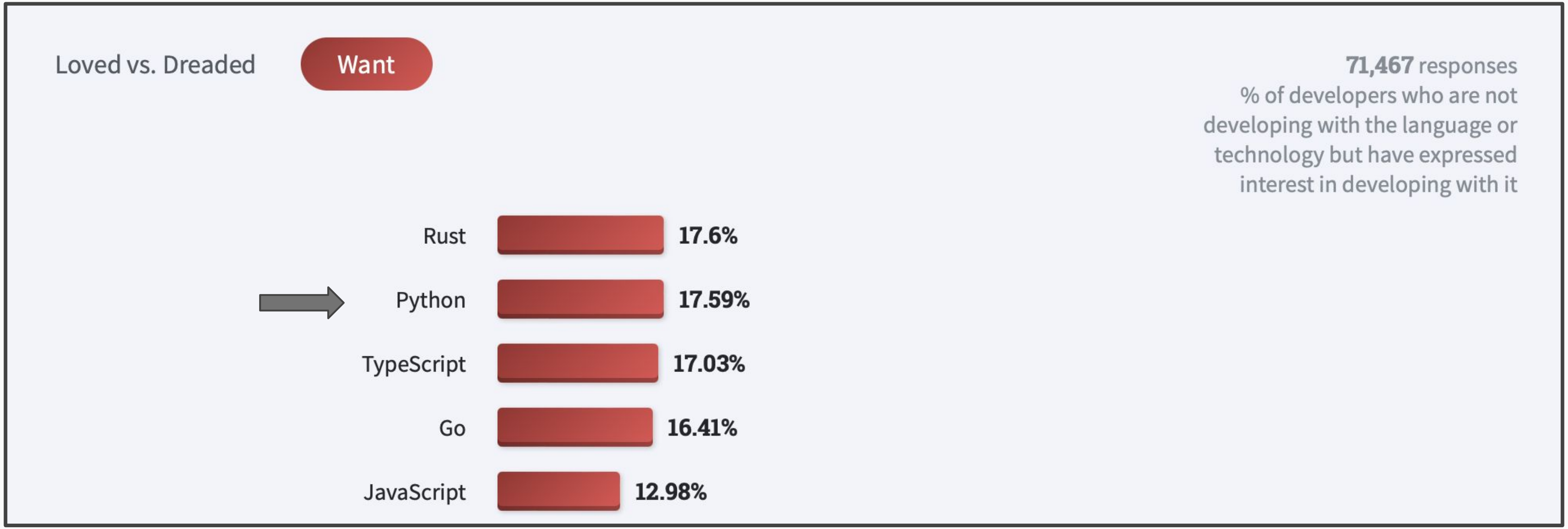


Motivation - Why Python?

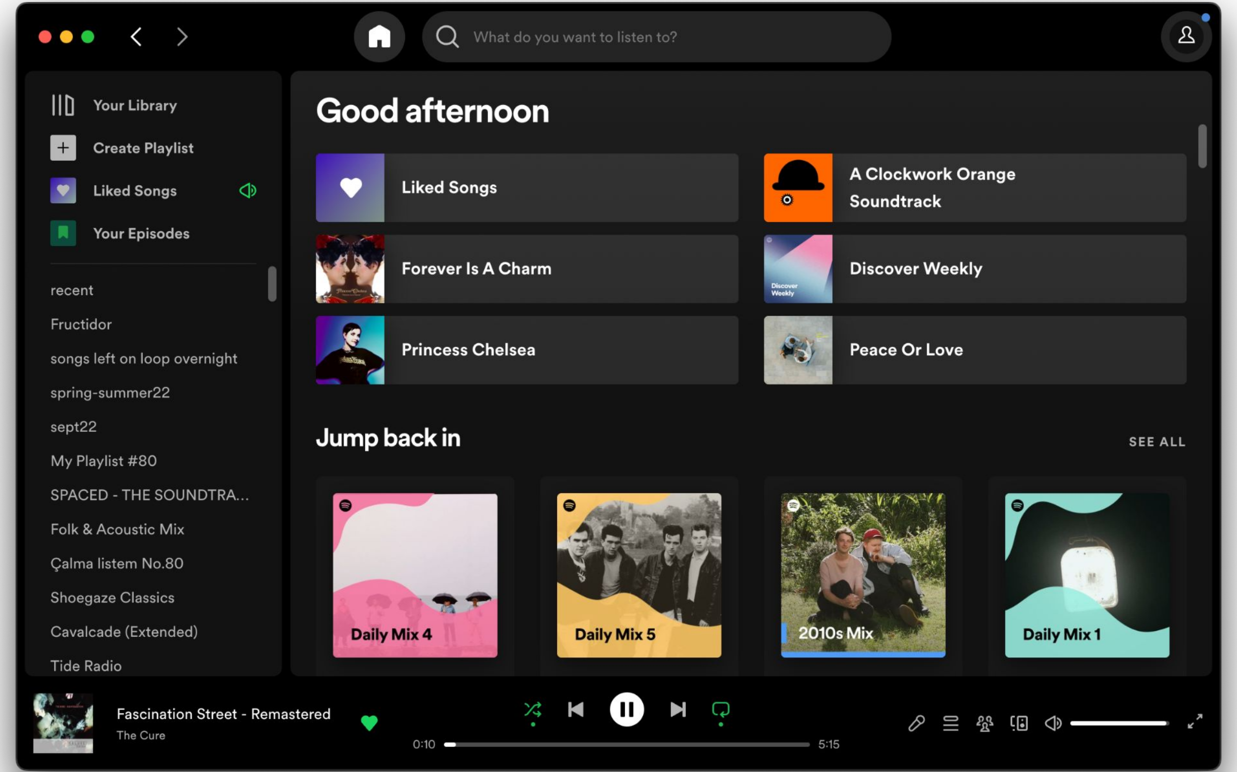
Which other frameworks and libraries have you done extensive development work in over the past year, and which do you want to work in over the next year?



Motivation - Why Python?

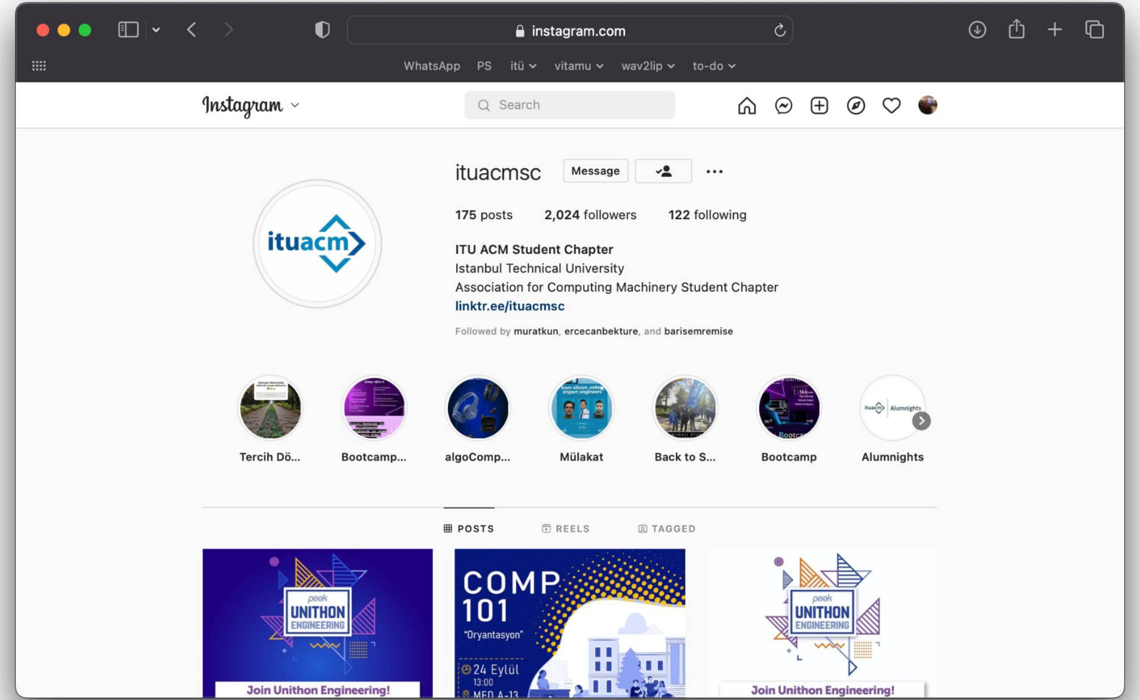


Developed in Python: Spotify



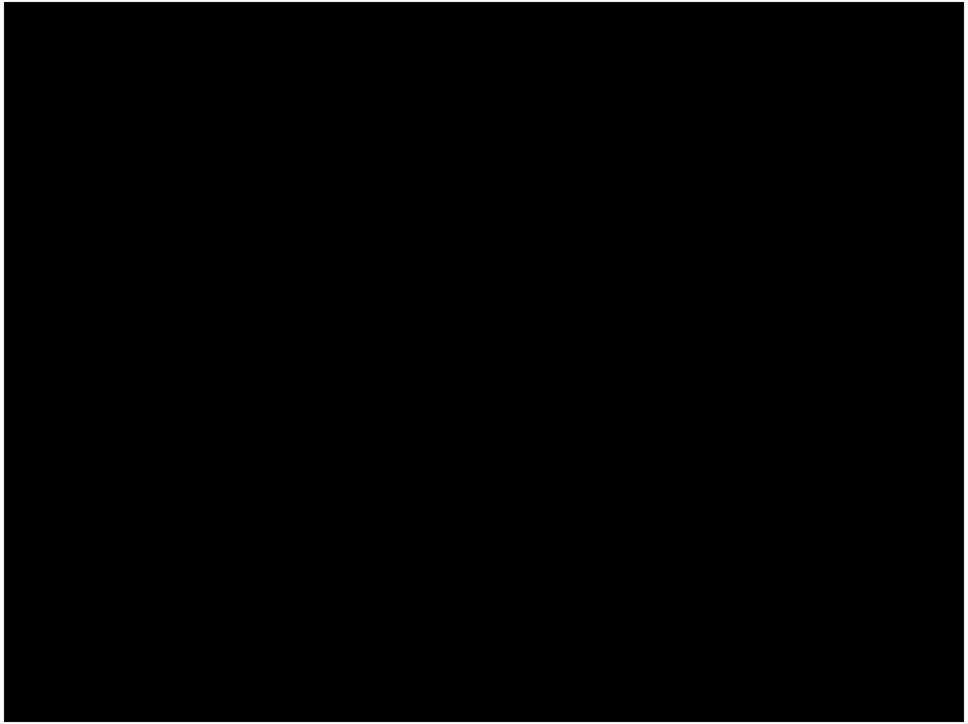
https://engineering.atspotify.com/2013/03/how-we-use-python-at-spotify/

Developed in Python: Instagram



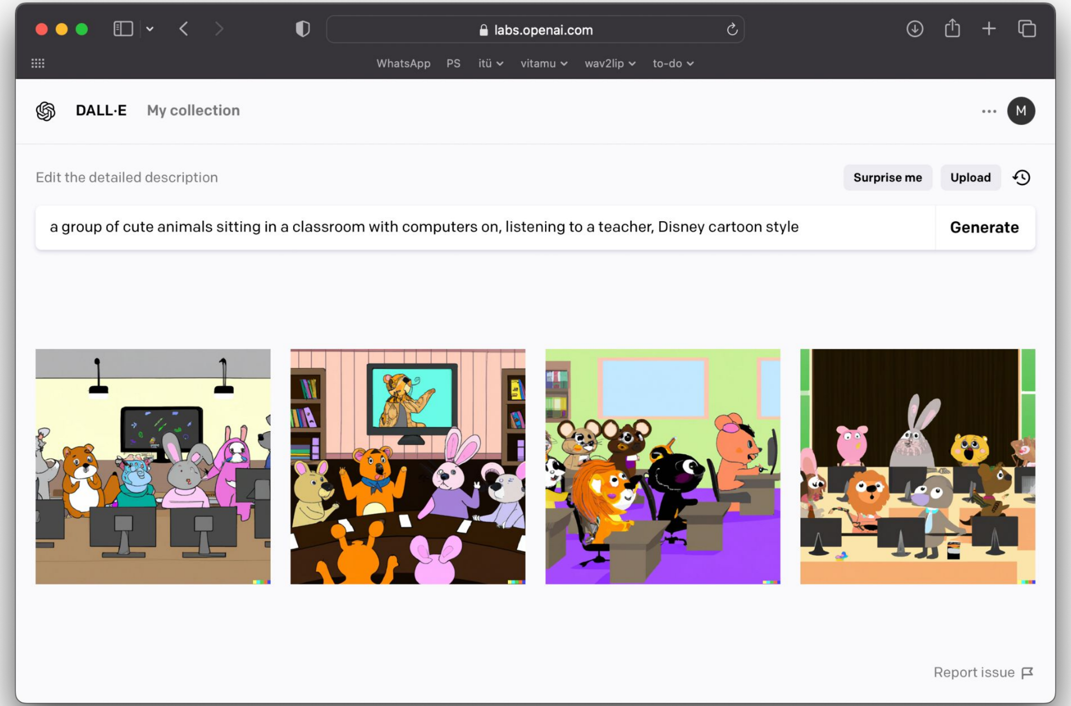
https://instagram-engineering.com/types-for-python-http-apis-an-instagram-story-d3c3a207fdb7

Developed in Python: Autonomous driving



https://github.com/eriklindernoren/PyTorch-YOLOv3

Developed in Python: AI Art



https://pypi.org/project/Python-DALLE/

What is a computer?

What is a computer?

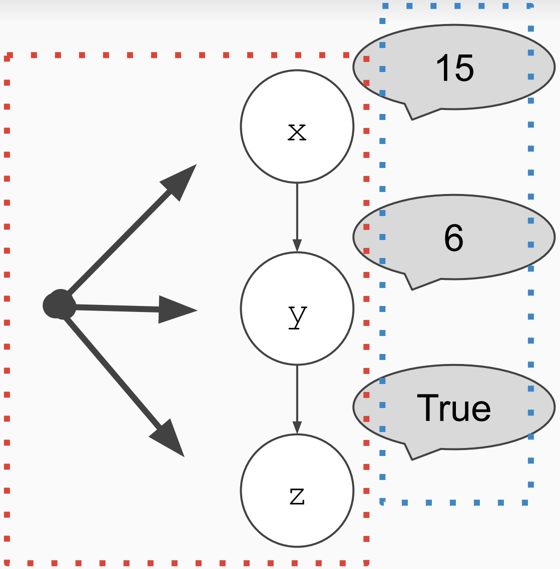
"A general purpose machine which can be programmed to carry out a finite set of arithmetic or logical operations."

- *a finite set of arithmetic or logical operations* -> performs computations
- *programmed* -> computations can be composed
- *general purpose* -> can be used for multiple aims

A simple computer program

```
x ← 5 * 3
y ← 4 + 2
z ← (x > y)
```

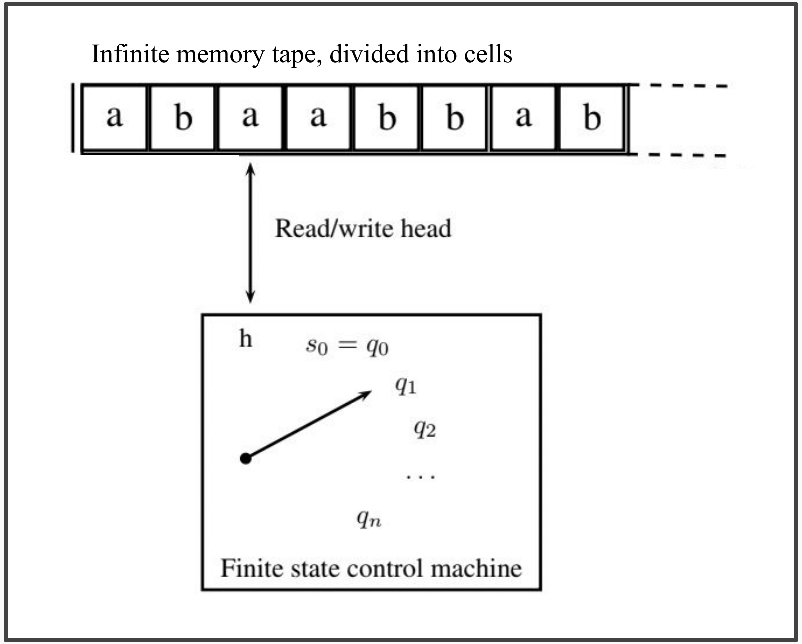
Program



Memory

Control machine

The abstract model of a computer



Turing Machine (1936)

capable of implementing every computer algorithm

not necessarily used in the modern computers (see Von Neumann architecture, 1945)

What is a program?

*a **precise description** of the process the machine will carry out*

which means:

anything that can be done by a computer can be precisely described

Q: What processes can a computer carry out?

A: performing **calculations**, saving & loading results to **variables**, making decisions based on their values (**control flow**).

Declarative vs imperative statements

Declarative statements define the values of variables directly.

```
y = 4
x = 2 * y
```

Imperative statements define the steps to achieve a value.

```
x = 0
y = 0
while y is less than 4:
     increment x by 2
     increment y by 1
```

How a program works

```
x = 0
y = 0
while y is less than 4:
    increment x by 2
    increment y by 1

```

Control fetches the first **line of instruction** from code memory

The instruction is **processed** by the computer

The result is **stored** in memory

Fundamental operations

Fundamental set of operations: Assignment

Assignment: The equals sign (=) computes the value on right hand side and gives it the name on the left hand side.

Code	Variable Name	Value
`var1 = 10`	var1	10
`var2 = 3 * 5`	var2	15
`var3 = var1 - var2`	var3	-5

Fundamental set of operations: Comparisons

Comparison operators: Less than (<), greater than (>), equality check (==), not equals (!=), less than or equal (<=), greater than or equal (>=)

	Variable Name	Operands	Value
`r1 = 5 == 10`	r1	(5), (10)	False
`r2 = var1 < var2`	r2	var1 (10), var2 (15)	True
`r3 = var3 == var1 - var2`	r3	var3 (-5), var1 - var2 (-5)	True

Assignment vs **Equality Check**

Symbol: **=**

Takes a variable and a value:
variable = value

Assigns value to variable, used as **a <u>statement</u> on its own**

Calculates the **left hand side** and gives it the name of the variable on **right hand side**

Symbol: **==**

Takes two operands:
a == b

Produces a **value**, used on **left hand side** as an **<u>expression</u>**

Evaluates to **True** or **False**

More on assignments

Assignments are **NOT permanent**.

When you make an assignment, the value is **bound** to that name.

You can assign another value to the same variable, and the latest value will be bound to that name.

```
var1 = 2*4
var1 = 7
endOfProgram
```

Value of `var1`**:** 7

The value of 2*4 is not retained!

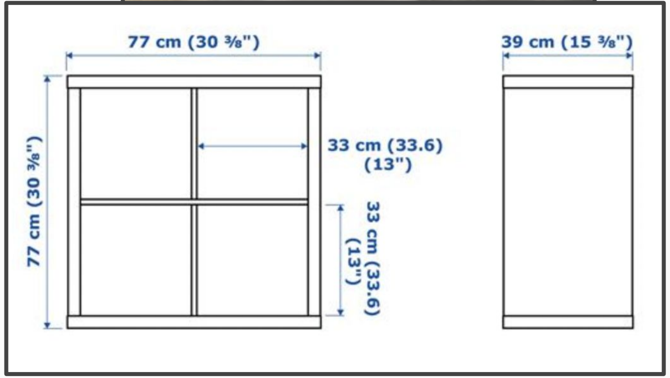
Fundamental set of operations: Arithmetics

Mathematical operators: Addition (+), subtraction (-), multiplication (*), division (/), exponent (**), modulus (%)

	Math	Value		Math	Value
`var1 = 5 + 3`	$5 + 3$	8	`var4 = 6 / 3`	$6 \div 2$	2
`var2 = 12 - 5`	$12 - 5$	7	`var5 = 3 ** 2`	3^2	9
`var3 = 4 * 9`	4×9	36	`var6 = 10 % 3`	$10 \ (mod\ 3)$	1

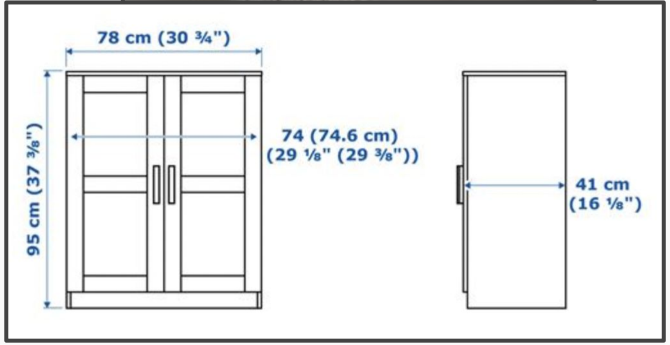
Example program: Volume of a furniture

```
widthA = 77
heightA = 77
areaA = widthA * heightA
depthA = 39
volA = areaA * depthA
...
```



Example program: Volume of a furniture

```
widthB = 78
heightB = 95
areaB = widthB * heightB
depthB = 41
volB = areaB * depthB
isALarger = volA > volB
```



Fundamental set of operations: Branching

Branching: Move the control system based on computations

```
12 if isALarger: go to 15
13 totalPrice = priceOfB
14 go to 16
15 totalPrice = priceOfA
16 ...
```

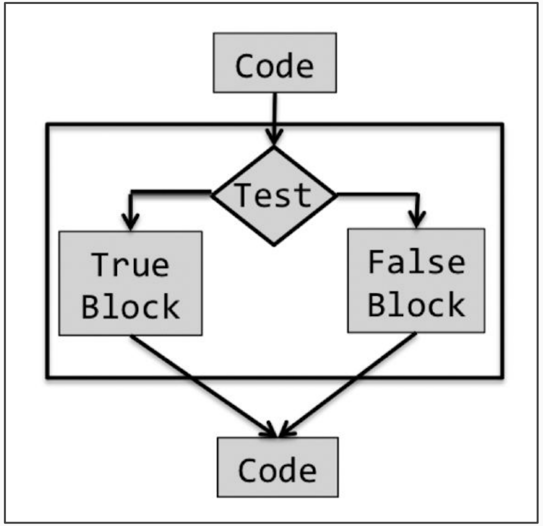
We can skip lines, based on the value of a variable (isALarger).

If isALarger is **True**, control skips from L12 to L15 and L13-14 are skipped.

If isALarger is **False**, control skips from L14 to L16 and L15 is skipped.

Fundamental set of operations: Conditionals

```
Code
if test:
    TrueBlock
else:
    FalseBlock
Code
```



Fundamental set of operations: Looping

Branching can be used to repeat a code piece.

```
16 tax = totalPrice * 0.1
17 totalPrice = totalPrice += tax
18 if totalPrice < 100: go to 16
19 endProgram
```

Instead of skipping, we can move the control head to a previous location.

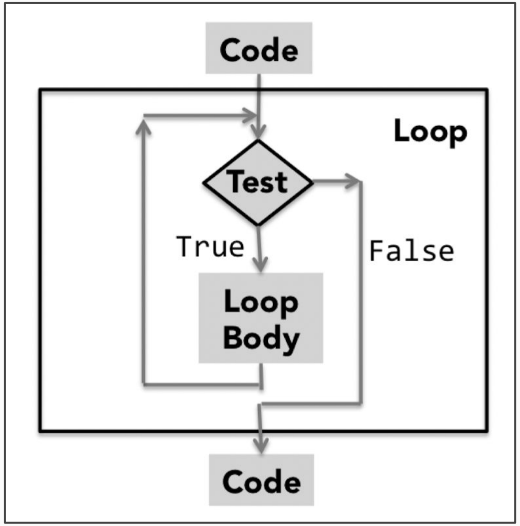
In this example, L16-L17 runs, L18 runs a comparison, and L16-L17 repeats until the comparison becomes false.

Fundamental set of operations: Looping

```
Code
while test:
    LoopBody



Code
```



Summary

- Our computer programs **in imperative paradigm** will describe the computations the computer will need to carry **step-by-step**.
- We will model real world scenarios by dividing them to use the **finite instruction capabilities** of the computer.
- We can perform **arithmetic** and **logical** operations, obtain **values**, store them in **variables.**

Summary

- We can use the values in variables to **branch to** different lines of instructions, allowing us to
  - **make decisions**
  - **skip lines**
  - **repeat sections of code**

Additional operations

Additional operations: Subroutines

In the looping example, we saw that a piece of code can be repeated multiple times. We can also give these pieces a **name** and use them as **subroutines**.

This is similar to assigning a value to a variable, but now, we are assigning a set of instructions to a variable to run them later.

```
if totalPrice < 100: go to Routine
endProgram
```

```
label Routine
totalPrice = totalPrice += tax
returnBack
```

Additional operations: Input Output (I/O)

Input: Allow computer user to enter values for variables.

Output: Write the values of variables on screen.

	Action	Result
`var1 = input()`	User enters a number and presses ‘Enter’.	The number from user is assigned to `var1`.
`print(var1)`	The symbol for number assigned to var1 is shown on screen.	No value is changed.

Additional operations: Text representation

In computers, we use **numbers** for inner representation of data. All the computations are done on numerical values.

However, we can decide on arbitrary rules to **transform numbers to symbols**, allowing us to use letters and text. These are called **strings**.

If we store the ***type*** of a variable (e.g. number, string) along with its value, the computer can transform our numbers to letters. (*Assume that we have a subroutine **string** that knows to transform from numbers to letters.)*

Additional operations: Text representation

```
text = string(72, 69, 76, 76, 79)
print(text)
```

Program

```
HELLO
```

Output seen by user

```
label string
text += letterValueOf(number)
returnBack text
```

Mysterious string routine

Text representation with ASCII encoding

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	*[NULL]*	32	20	*[SPACE]*	64	40	**@**	96	60	**`**
1	1	*[START OF HEADING]*	33	21	**!**	65	41	**A**	97	61	**a**
2	2	*[START OF TEXT]*	34	22	**"**	66	42	**B**	98	62	**b**
3	3	*[END OF TEXT]*	35	23	**#**	67	43	**C**	99	63	**c**
4	4	*[END OF TRANSMISSION]*	36	24	**$**	68	44	**D**	100	64	**d**
5	5	*[ENQUIRY]*	37	25	**%**	69	45	**E**	101	65	**e**
6	6	*[ACKNOWLEDGE]*	38	26	**&**	70	46	**F**	102	66	**f**
7	7	*[BELL]*	39	27	**'**	71	47	**G**	103	67	**g**
8	8	*[BACKSPACE]*	40	28	**(**	72	48	**H**	104	68	**h**
9	9	*[HORIZONTAL TAB]*	41	29	**)**	73	49	**I**	105	69	**i**
10	A	*[LINE FEED]*	42	2A	*****	74	4A	**J**	106	6A	**j**
11	B	*[VERTICAL TAB]*	43	2B	**+**	75	4B	**K**	107	6B	**k**
12	C	*[FORM FEED]*	44	2C	**,**	76	4C	**L**	108	6C	**l**
13	D	*[CARRIAGE RETURN]*	45	2D	**-**	77	4D	**M**	109	6D	**m**
14	E	*[SHIFT OUT]*	46	2E	**.**	78	4E	**N**	110	6E	**n**
15	F	*[SHIFT IN]*	47	2F	**/**	79	4F	**O**	111	6F	**o**
16	10	*[DATA LINK ESCAPE]*	48	30	**0**	80	50	**P**	112	70	**p**
17	11	*[DEVICE CONTROL 1]*	49	31	**1**	81	51	**Q**	113	71	**q**
18	12	*[DEVICE CONTROL 2]*	50	32	**2**	82	52	**R**	114	72	**r**
19	13	*[DEVICE CONTROL 3]*	51	33	**3**	83	53	**S**	115	73	**s**
20	14	*[DEVICE CONTROL 4]*	52	34	**4**	84	54	**T**	116	74	**t**
21	15	*[NEGATIVE ACKNOWLEDGE]*	53	35	**5**	85	55	**U**	117	75	**u**
22	16	*[SYNCHRONOUS IDLE]*	54	36	**6**	86	56	**V**	118	76	**v**
23	17	*[ENG OF TRANS. BLOCK]*	55	37	**7**	87	57	**W**	119	77	**w**
24	18	*[CANCEL]*	56	38	**8**	88	58	**X**	120	78	**x**
25	19	*[END OF MEDIUM]*	57	39	**9**	89	59	**Y**	121	79	**y**
26	1A	*[SUBSTITUTE]*	58	3A	**:**	90	5A	**Z**	122	7A	**z**
27	1B	*[ESCAPE]*	59	3B	**;**	91	5B	**[**	123	7B	**{**
28	1C	*[FILE SEPARATOR]*	60	3C	**<**	92	5C	****	124	7C	**|**
29	1D	*[GROUP SEPARATOR]*	61	3D	**=**	93	5D	**]**	125	7D	**}**
30	1E	*[RECORD SEPARATOR]*	62	3E	**>**	94	5E	**^**	126	7E	**~**
31	1F	*[UNIT SEPARATOR]*	63	3F	**?**	95	5F	**_**	127	7F	*[DEL]*

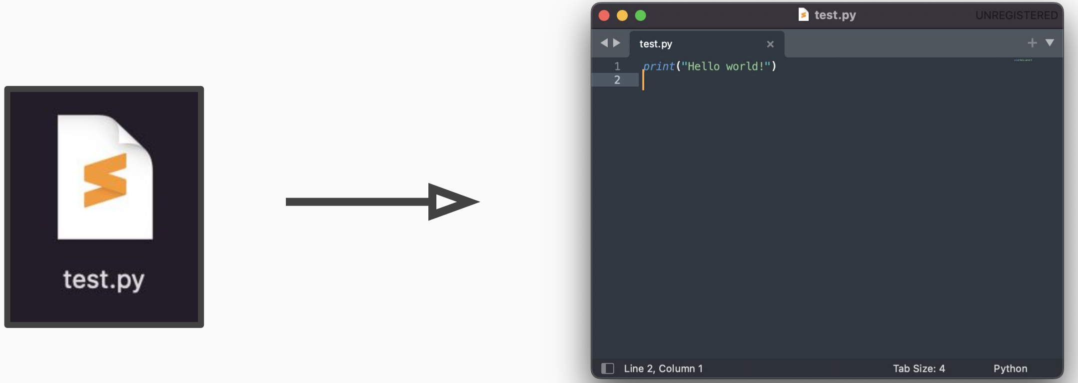
https://en.wikipedia.org/wiki/ASCII

Coding in Python

Coding in Python

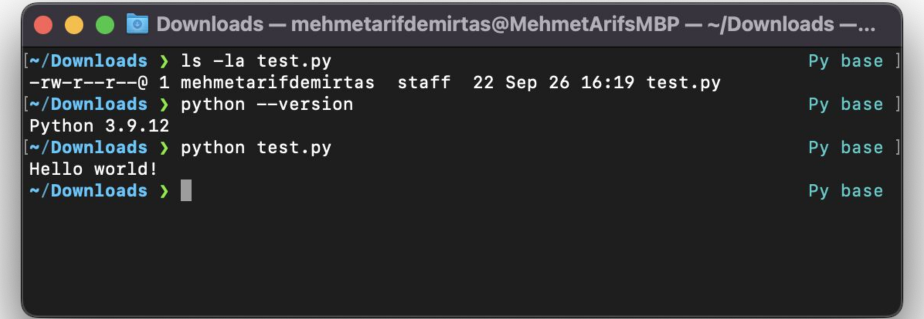
In Python, the instructions we have seen can be stored in:

- *scripts.* A Python script is a text file with the extension *.py and it contains the lines of the program.



Coding in Python

The Python scripts are **run with Python interpreter**, which is the tool that will read your code line-by-line and transform each line to **machine language** for computer to execute them. You can start the interpreter in **shell**.



Coding in Python

In Python, we can also write instructions **directly into a REPL session.**

REPL stands for **Read, Evaluate, Print Loop**.

When you start a REPL session from shell, you can write your instructions directly, a line at a time, and the interpreter runs and prints the results instantly.

Coding in REPL

```
~/Downloads ❯ python                                    Py base
Python 3.9.12 (main, Apr  5 2022, 01:53:17)
[Clang 12.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license"
 for more information.
>>> x = 5
>>> print(x + 3)
8
>>> x = 2
>>> print(x + 3)
5
>>> 
```

Shell prompt

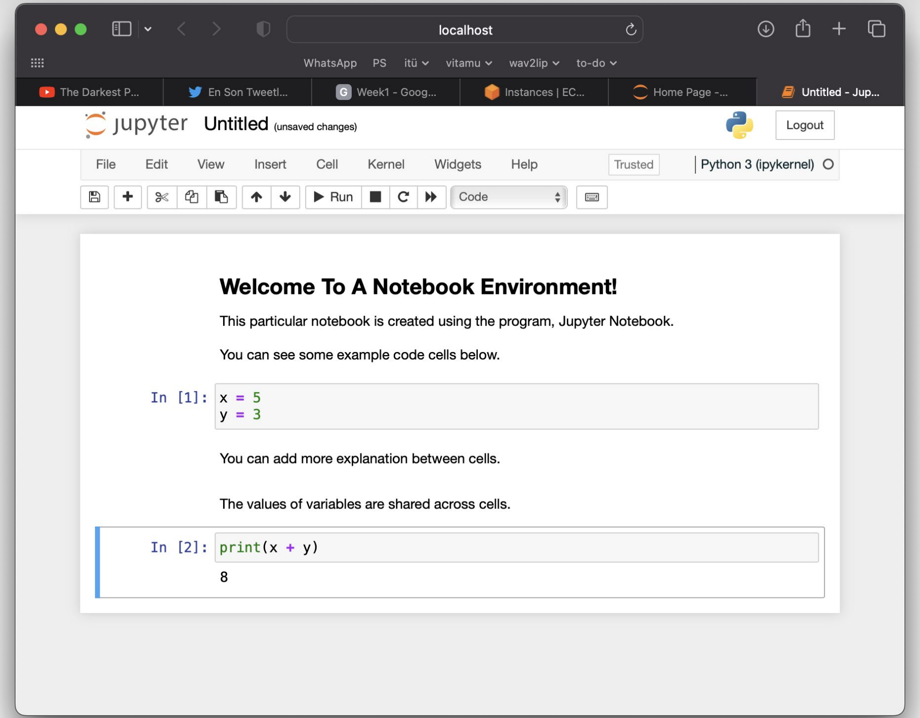
Coding in Python

Finally, Python code can be also written in **notebooks.**

A notebook is a document (with the extension .ipynb) that contains both text content and **code cells**. The statements can be written into code cells and each cell can be run independently. When you start a notebook, a **kernel** is associated and the values of the variables are shared across the kernel.

Therefore, you can separate your code into cells and explain them with text in between.

Coding in Notebooks



Next week

- Types of data representations in Python
- More on strings
- Common subroutines (functions) implemented in Python
- Conditional structures in Python
- Loops and control flow on a higher level

References

- Minsky, M. L. (1967). Computation. Englewood Cliffs: Prentice-Hall.
- Guttag, J. V. (2016). Introduction to computation and programming using Python: With application to understanding data. MIT Press.

- ITU BLG311E Formal Languages and Automata course slides, Tolga Ovatman & Berk Canberk, 2020
- MIT 6.0001 Introduction to Computer Science and Programming in Python course slides, Ana Bell, 2016