# **Communicate**

**Where are we now? What have we seen/done/studied?**

* Data cleaning
* Code organization
  + Moving from notebooks to Python Classes
  + Combining Classes
  + Exploratory Data Analysis
  + Merging tables
  + etc.
* Multivariate Linear Regression
* Logistic Regression

**What's next?**

* [Cost-Benefit Analysis](https://www.investopedia.com/terms/c/cost-benefitanalysis.asp)
* Summarize your findings
* Communicate

## **1. Why Communication?**

Try to answer this typical interview question for Data Science/Analytics jobs:

“Tell me about a project where you *actually* changed the course of action of your business because of your data."

Here, proper communication is **very** important!

Let's look at how we can:

* Influence! 👉 **Data Science as U-Turn (= using data to influence directions)**
* Collaborate
* Write well-documented code to ensure it can be replicated

## **2. Qualities of a Well-Communicated DS Project**

#### **Make the complex easy to understand**

What is well understood is clearly said, And the words to say it flow with ease.

*Original: Ce que l'on conçoit bien, s'énonce clairement, Et les mots pour le dire arrivent aisément.*

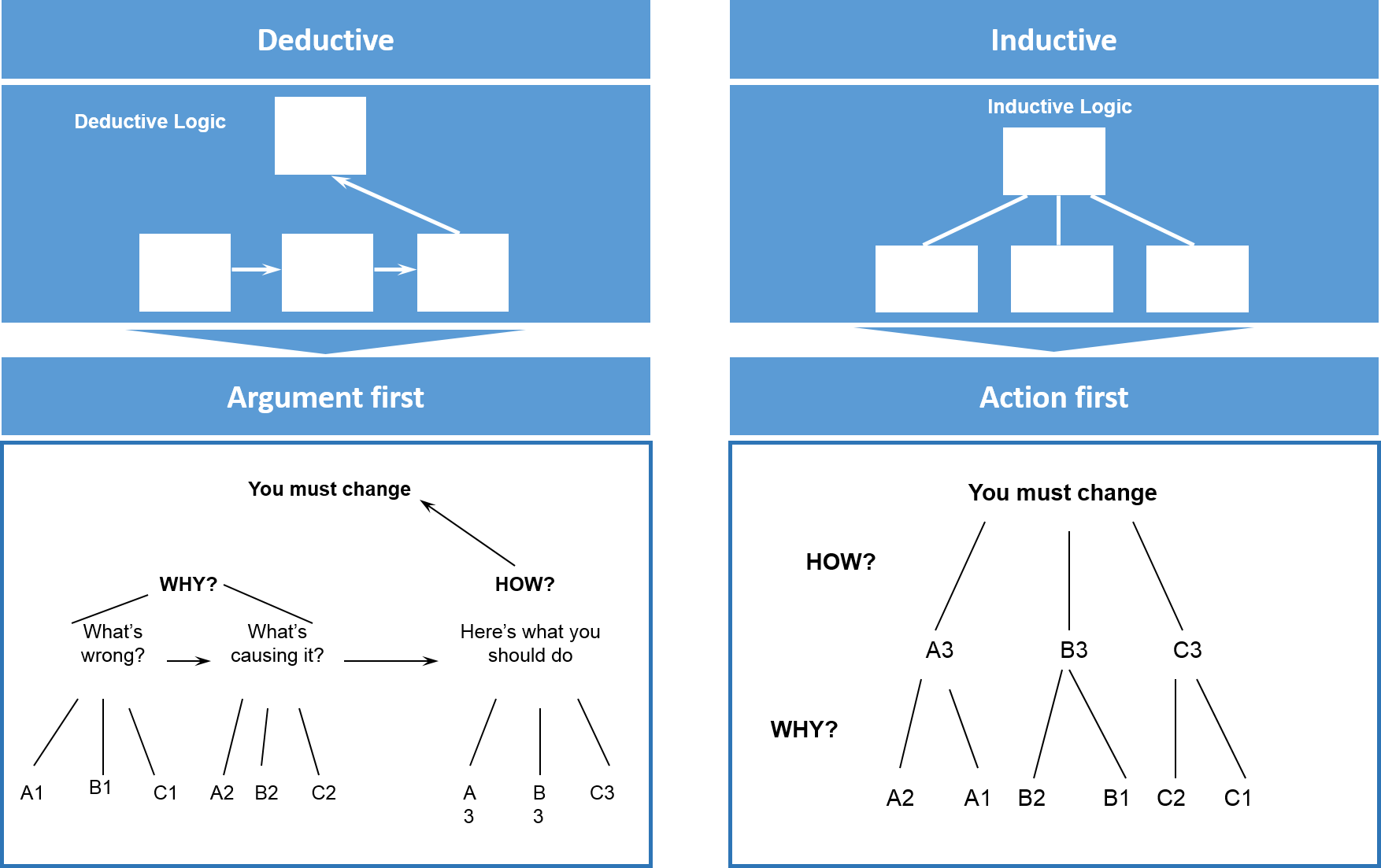
🇫🇷 [Nicolas Boileau-Despréaux](https://en.wikipedia.org/wiki/Nicolas_Boileau-Despr%C3%A9aux), L'Art poétique (1674)

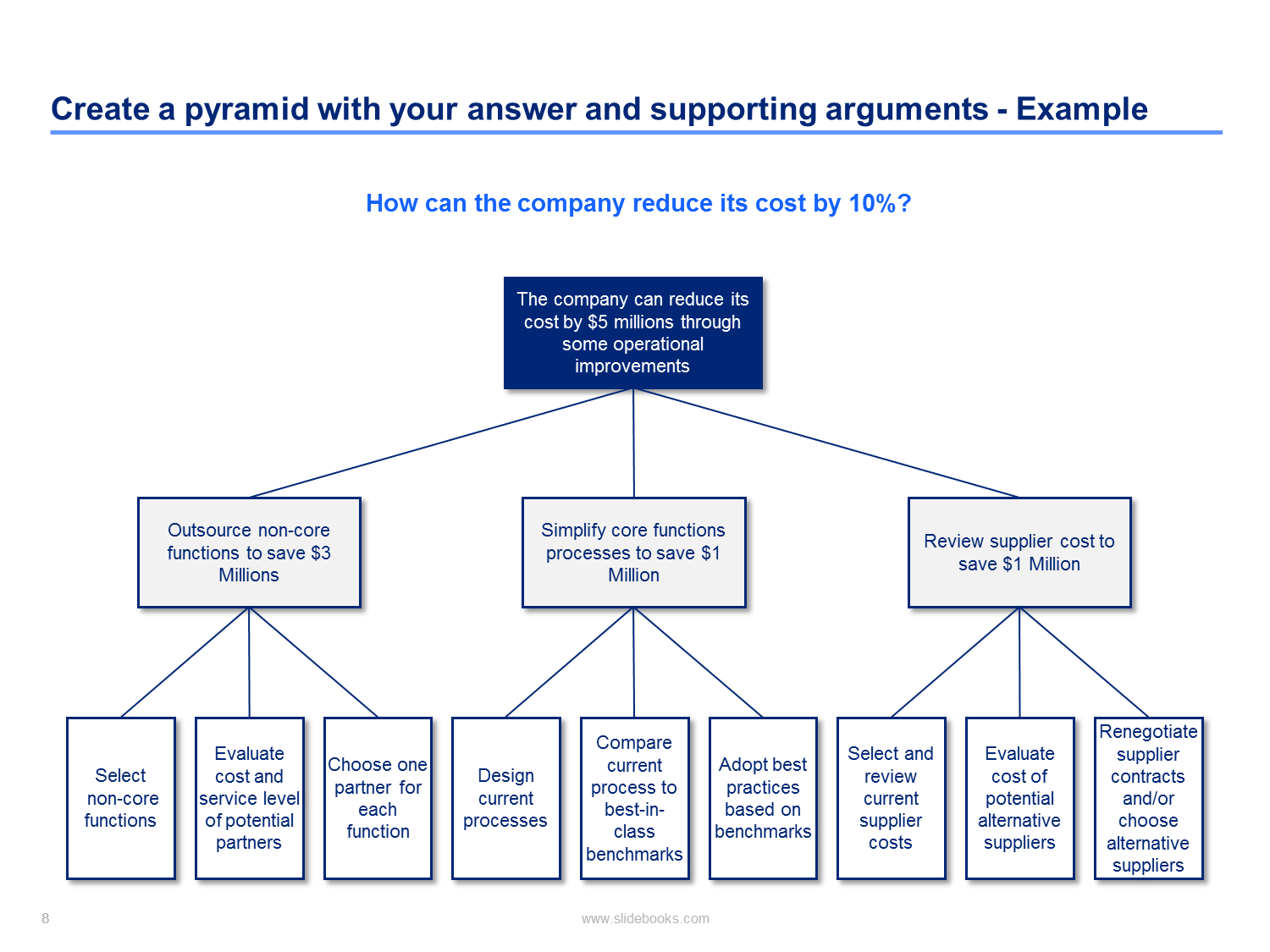
#### **⚡️ Be Concise!**

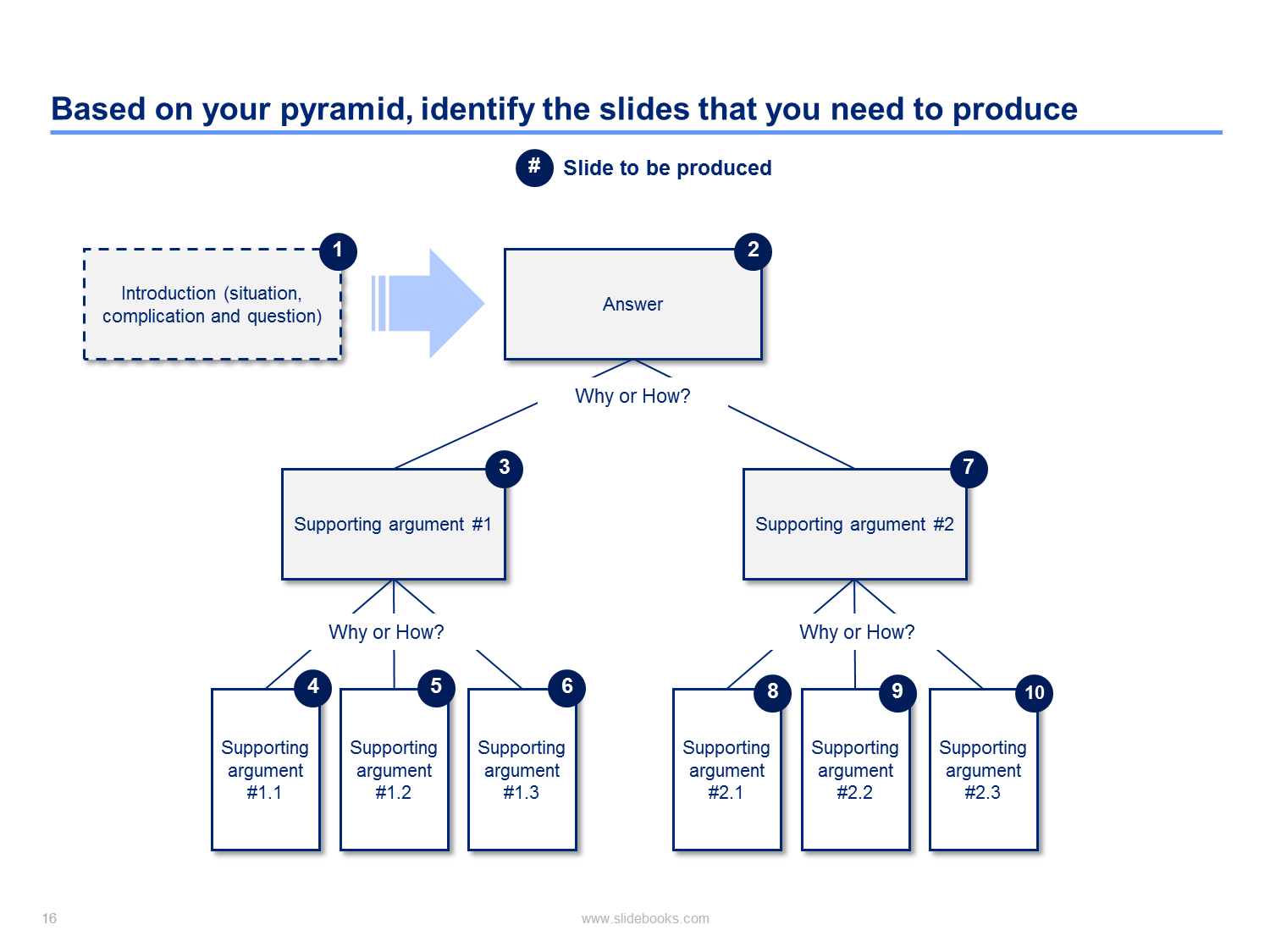
#### **Follow the** [**Pyramid Principle**](https://gettingbettereveryday.org/2018/10/05/what-you-could-learn-from-barbra-mintos-the-pyramid-principle-2009-172-pages/)

👉 Action/Answer first, explain why after!

👉 More efficient under time constraints







## **3. Understand Trade-Offs**

### **Consider the following example**

Your **Product Manager (PM)** comes up with an idea to **increase the conversion rate** on a website

* This website has ~100K visitors per month
* On average, a visit provides a ~20 BRL benefit (100K \* 20 BRL = 2.000.000 BRL)
* You have a team of 3 engineers who cost 500 BRL/day each

👉 You don't know **how long** it will take to code this feature  
👉 You don't know the **lift** (increase) this potential feature could have on the conversion rate.

❓ How do you inform your product manager ❓

**Step 1️⃣ - Model Benefits**

We estimate the potential lifts between +1% and +4%

lift = np.arange(0.01, 0.05, 0.01)

benefits = pd.DataFrame(

{'lift': lift, 'benefit': 100000 \* 20 \* lift}

)

benefits

|  | **lift** | **benefit** |
| --- | --- | --- |
| **0** | 0.01 | 20000.0 |
| **1** | 0.02 | 40000.0 |
| **2** | 0.03 | 60000.0 |
| **3** | 0.04 | 80000.0 |

**Step 2️⃣ - Model Costs**

* f
* (number of workdays of engineering team)
* Estimate between 20 and 50 days

workdays = np.arange(20, 60, 10)

costs = pd.DataFrame(

{'workdays': workdays, 'cost': workdays \* 500 \* 3}

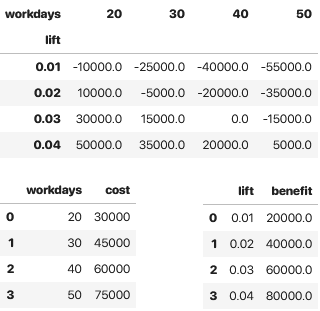
)

costs

|  | **workdays** | **cost** |
| --- | --- | --- |
| **0** | 20 | 30000 |
| **1** | 30 | 45000 |
| **2** | 40 | 60000 |
| **3** | 50 | 75000 |

**Step 3️⃣ - Sensitivity Matrix**

We can highlight results in a **sensitivity matrix** to see the **trade-off**:



**Step 3.1**

We have to create the **cartesian product** of both the cost and the benefit DataFrames

sensitivity = costs.merge(benefits, how='cross')

sensitivity

|  | **workdays** | **cost** | **lift** | **benefit** |
| --- | --- | --- | --- | --- |
| **0** | 20 | 30000 | 0.01 | 20000.0 |
| **1** | 20 | 30000 | 0.02 | 40000.0 |
| **2** | 20 | 30000 | 0.03 | 60000.0 |
| **3** | 20 | 30000 | 0.04 | 80000.0 |
| **4** | 30 | 45000 | 0.01 | 20000.0 |
| **5** | 30 | 45000 | 0.02 | 40000.0 |
| **6** | 30 | 45000 | 0.03 | 60000.0 |
| **7** | 30 | 45000 | 0.04 | 80000.0 |
| **8** | 40 | 60000 | 0.01 | 20000.0 |
| **9** | 40 | 60000 | 0.02 | 40000.0 |
| **10** | 40 | 60000 | 0.03 | 60000.0 |
| **11** | 40 | 60000 | 0.04 | 80000.0 |
| **12** | 50 | 75000 | 0.01 | 20000.0 |
| **13** | 50 | 75000 | 0.02 | 40000.0 |
| **14** | 50 | 75000 | 0.03 | 60000.0 |
| **15** | 50 | 75000 | 0.04 | 80000.0 |

**Step 3.2**

We can now add net\_profit and clean up the sensitivity matrix

sensitivity['net\_profit'] = sensitivity['benefit'] - sensitivity['cost']

sensitivity.drop(columns=['cost', 'benefit'], inplace=**True**) *# Dropping columns we don't need*

sensitivity.head()

|  | **workdays** | **lift** | **net\_profit** |
| --- | --- | --- | --- |
| **0** | 20 | 0.01 | -10000.0 |
| **1** | 20 | 0.02 | 10000.0 |
| **2** | 20 | 0.03 | 30000.0 |
| **3** | 20 | 0.04 | 50000.0 |
| **4** | 30 | 0.01 | -25000.0 |

*# Pivot is the easiest way to do it*

sensitivity.pivot(index='lift', columns='workdays', values='net\_profit')

| **workdays** | **20** | **30** | **40** | **50** |
| --- | --- | --- | --- | --- |
| **lift** |  |  |  |  |
| **0.01** | -10000.0 | -25000.0 | -40000.0 | -55000.0 |
| **0.02** | 10000.0 | -5000.0 | -20000.0 | -35000.0 |
| **0.03** | 30000.0 | 15000.0 | 0.0 | -15000.0 |
| **0.04** | 50000.0 | 35000.0 | 20000.0 | 5000.0 |

*# Creating a double index and unstacking also works!*

sensitivity.set\_index(['lift', 'workdays']).unstack()

|  | **net\_profit** | | | |
| --- | --- | --- | --- | --- |
| **workdays** | **20** | **30** | **40** | **50** |
| **lift** |  |  |  |  |
| **0.01** | -10000.0 | -25000.0 | -40000.0 | -55000.0 |
| **0.02** | 10000.0 | -5000.0 | -20000.0 | -35000.0 |
| **0.03** | 30000.0 | 15000.0 | 0.0 | -15000.0 |
| **0.04** | 50000.0 | 35000.0 | 20000.0 | 5000.0 |

*# The underscore used here is a useful way to grab the last output*

\_.style.map(**lambda** x : 'color: red' **if** x < 0 **else** 'color: black')

|  | **net\_profit** | | | |
| --- | --- | --- | --- | --- |
| **workdays** | **20** | **30** | **40** | **50** |
| **lift** |  |  |  |  |
| **0.010000** | -10000.000000 | -25000.000000 | -40000.000000 | -55000.000000 |
| **0.020000** | 10000.000000 | -5000.000000 | -20000.000000 | -35000.000000 |
| **0.030000** | 30000.000000 | 15000.000000 | 0.000000 | -15000.000000 |
| **0.040000** | 50000.000000 | 35000.000000 | 20000.000000 | 5000.000000 |

## **4. Powerful Forms of Communication**

1. Docs and code itself (see [Communication Through Code](https://medium.com/the-mighty-programmer/code-communication-ac4a8a4a4f9a))
2. Interactive tools
3. Notebooks

### **4.1 - Code and Docs**

#### **Write the Docs**

****

* [Write the Docs](https://www.writethedocs.org/) is a community dedicated to the art and science of technical documentation
* Why good technical documentation?
  + Your code in 6 months will likely belong to someone else
  + Well-documented code helps others understand what it does
  + Detailing your methodology makes it easy for others to follow it
  + If you come back to undocumented code, you have no idea what you did 🫠  
    *Even just a week after writing it!* 🙃

#### **args and kwargs**

**def** foo(required, \*args, \*\*kwargs):

*"""This is the function docstring"""*

print(required)

**if** args:

print(args)

**if** kwargs:

print(kwargs)

foo(1, 2, 3, bar = 5)

1

(2, 3)

{'bar': 5}

a\_list = [1, 2, 3]

a\_dict = {'bar': 5}

foo(\*a\_list, \*\*a\_dict)

1

(2, 3)

{'bar': 5}

**class** **Student**:

school = 'lewagon'

**def** \_\_init\_\_(self, name, age):

self.name = name

self.age = age

alice = Student('alice', 20)

alice.\_\_dict\_\_

{'name': 'alice', 'age': 20}

d = {'a': 1, 'b': 2}

d.update(c = 3)

d

{'a': 1, 'b': 2, 'c': 3}

*# Student class with allowed optional attributes*

**class** **Student**:

school = 'lewagon'

**def** \_\_init\_\_(self, name, age, \*\*kwargs):

self.name = name

self.age = age

self.\_\_dict\_\_.update(\*\*kwargs)

alice = Student('alice', 20, nationality='brazilian')

alice.nationality

'brazilian'

alice.\_\_dict\_\_

{'name': 'alice', 'age': 20, 'nationality': 'brazilian'}

*# DataStudent child class with abstract arguments*

**class** **DataStudent**(Student):

course = 'data'

*# CAREFUL: the batch parameter should*

*# not be passed to the parent class!*

**def** \_\_init\_\_(self, name, age, batch, \*\*kwargs):

super().\_\_init\_\_(name, age, \*\*kwargs)

self.batch = batch *# now we can add a batch!*

DataStudent('alice', 20, '#1207', nationality='brazilian').\_\_dict\_\_

{'name': 'alice', 'age': 20, 'nationality': 'brazilian', 'batch': '#1207'}

#### **Typing Hints**

Recent versions of Python allow you to explicitly specify the **datatype** needed for both the function's inputs and outputs

[Real Python - Type Checking](https://realpython.com/python-type-checking/)

**def** say\_hi(name: str) -> str:

**return** name + ' says Hi!'

say\_hi('alice')

'alice says Hi!'

**def** cost\_of\_reviews(s: pd.Series) -> int:

**return** s.map({

1: 100,

2: 50,

3: 40,

4: 0,

5: 0

}).sum()

cost\_of\_reviews(pd.Series([1, 1, 5, 4]))

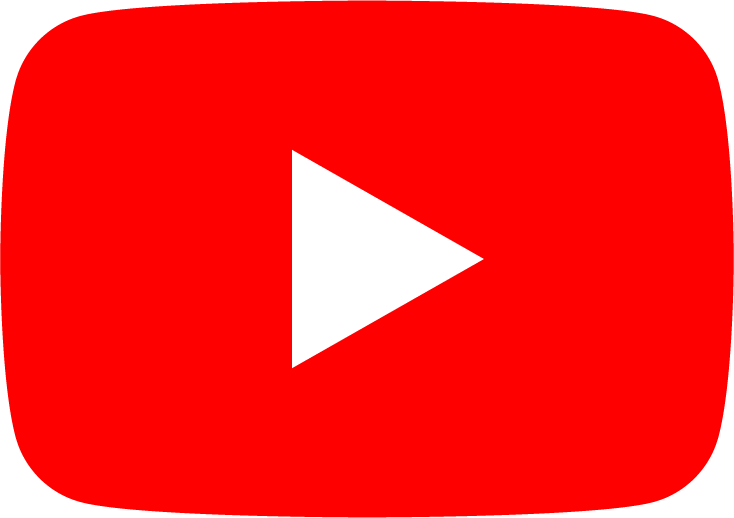
200

### **4.2 - Interactive Tools**

*# start at 6min30, stops at 9min17*

**from** **IPython.display** **import** HTML, IFrame

IFrame("http://www.youtube.com/embed/8QiPFmIMxFc?t=388", width="560", height="315")

[Bret Victor's "Inventing on Principle"](http://www.youtube.com/embed/8QiPFmIMxFc?t=388), on the need for interactivity and fast feedback loops in creative processes

#### **ipywidgets**

**from** **ipywidgets** **import** interact

@interact

**def** plot\_polynom(a=[0, 1, 2, 3], b=2):

x = np.arange(-10, 10, 0.1)

y = a\*x\*\*3+ b\*x\*\*2

plt.plot(x,y); plt.xlim(xmin=-10, xmax=10); plt.ylim(ymin=-100, ymax=100)

#### **🤔Python decorators @**

**def** my\_decorator(func):

**def** wrapper():

print("I'm before the method call")

func()

print("I'm after the method call")

**return** wrapper

**def** say():

print("Hi!")

say = my\_decorator(say)

say()

I'm before the method call

Hi!

I'm after the method call

**def** my\_decorator(func):

**def** wrapper():

print("I'm before the method call")

func()

print("I'm after the method call")

**return** wrapper

@my\_decorator

**def** say():

print("Hi!")

say()

I'm before the method call

Hi!

I'm after the method call

@my\_decorator is just an easier way of **wrapping** features around existing methods

* Easy to add (does not depend on your function name)
* Easy to remove (your function name does not stay modified after removal)

📚 For the curious among you, here is a [great blog post](https://realpython.com/primer-on-python-decorators/) on Python decorators (1h read)!

#### **Plotly Express**

**import** **plotly.express** **as** **px**

df = px.data.tips()

df

fig = px.scatter(df, x="size", y="tip", color="day")

fig.show()

### **4.3 - Notebook-Based Presentations**

**Step 1️⃣: Prepare a Clean Notebook**

* Enable selection of cell slide type
  + **Jupyter Notebook:** View → Right Sidebar → Show Notebook Tools → Common Tools
  + **Jupyter Lab:** View → Appearance → Show Right Sidebar
  + **VS Code:** Right-Click on a cell's status bar (bottom) → Switch Slide Type
* Select the slide type for each cell
  + Slide → main chapters
  + Sub-Slide → new slide below the main chapter
  + Fragment → subset of the slide directly above, appears when pressing the arrow down
  + - = display immediately after the cell above
  + Skip = does not display (e.g. hides code logic)
  + Notes = does not display, for private notes

**Step 2️⃣: Convert to a Static Slide-Based HTML Doc with Jupyter nbconvert**

jupyter nbconvert --to slides --post serve <your\_notebook.ipynb>

**Some Additional Tips**

* Use **Markdown** for...everything! 👉 [Markdown cheatsheet](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet)
* Use HTML for style and images 👉 [HTML cheatsheet](https://htmlcheatsheet.com/)
* Use
* L
* a
* T
* e
* X
* for math! 👉 [LaTeX cheatsheet](https://wch.github.io/latexsheet/latexsheet-a4.pdf)

👉 Use *emojis* (macOS: ⌘-⌃-Space, Windows: Win-.)

* Esc - M changes a cell to Markdown mode
* Esc - Y changes it back to code mode

**Remember:** if you want to share an interactive notebook, be careful with dependencies (imports/packages)

**Notebook Best Practices**

* Use Collapsible Headings and Table of Content
* Notebooks should be executable from top to bottom
* Name your variables carefully
* Use dummy names such as tmp or \_ when needed
* Clear useless variables from RAM when not needed (del my\_variable)
* Clear your code and merge cells when relevant (Shift-M)
* Hide your cell outputs to gain space (double-click on the red Out[]: section to the left of your cell).

## **Bibliography**

* 📚 [Thinking Fast and Slow](https://en.wikipedia.org/wiki/Thinking,_Fast_and_Slow)
* 📚 [The Pyramid Principle](https://gettingbettereveryday.org/2018/10/05/what-you-could-learn-from-barbra-mintos-the-pyramid-principle-2009-172-pages/)

## **Your Turn!**

🔥 Get ready for your presentations!