INTRO TO PYTHON

LEARNING OUTCOMES

LO1: master the basic **syntax rules** of Python
LO2: use **Jupyter Notebooks** to write Python code
LO3: apply a **debugging workflow**

AGENDA

1. What is Python?
2. Data types and variables
3. Methods
4. Iterable data types
5. Loops
6. Control flow
7. Debugging
8. Support & documentation

what is python?

t+f;

Python is a **programming language**

A programming language simply is the language people use to **speak to computers**

Some languages are used for specific tasks,
Python is the most **polyvalent** language

web development

web development

NETFLIX

game development

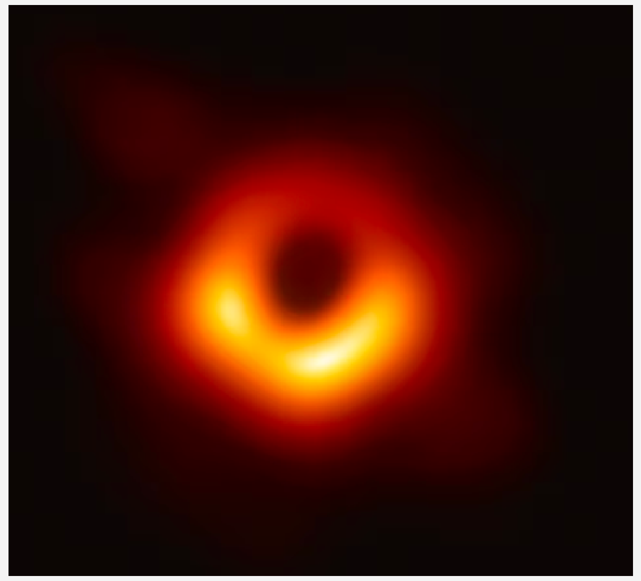
The SIMS 4

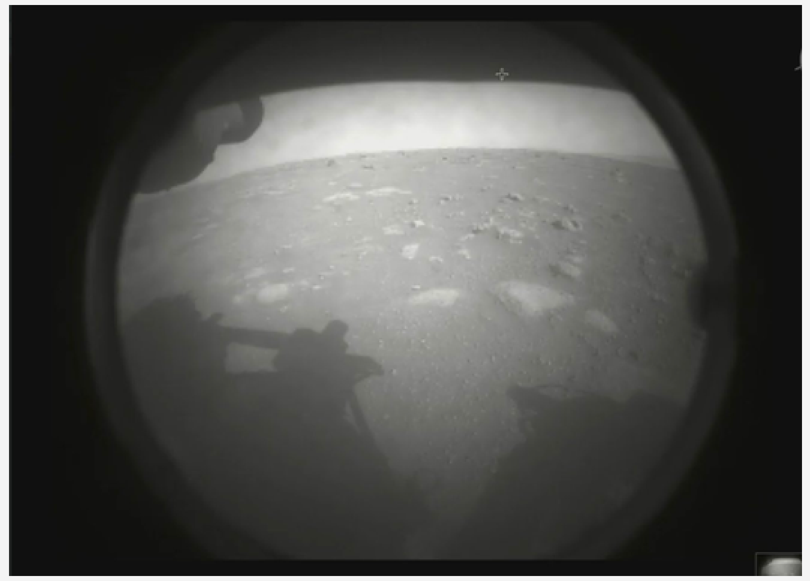
SID MEIER'S
CIVILIZATION IV

BATTLEFIELD 2

science & engineering

science & engineering





data science & AI

data science & AI

amazon

Uber

why learn how to code?

1

save time by automating mundane tasks

2

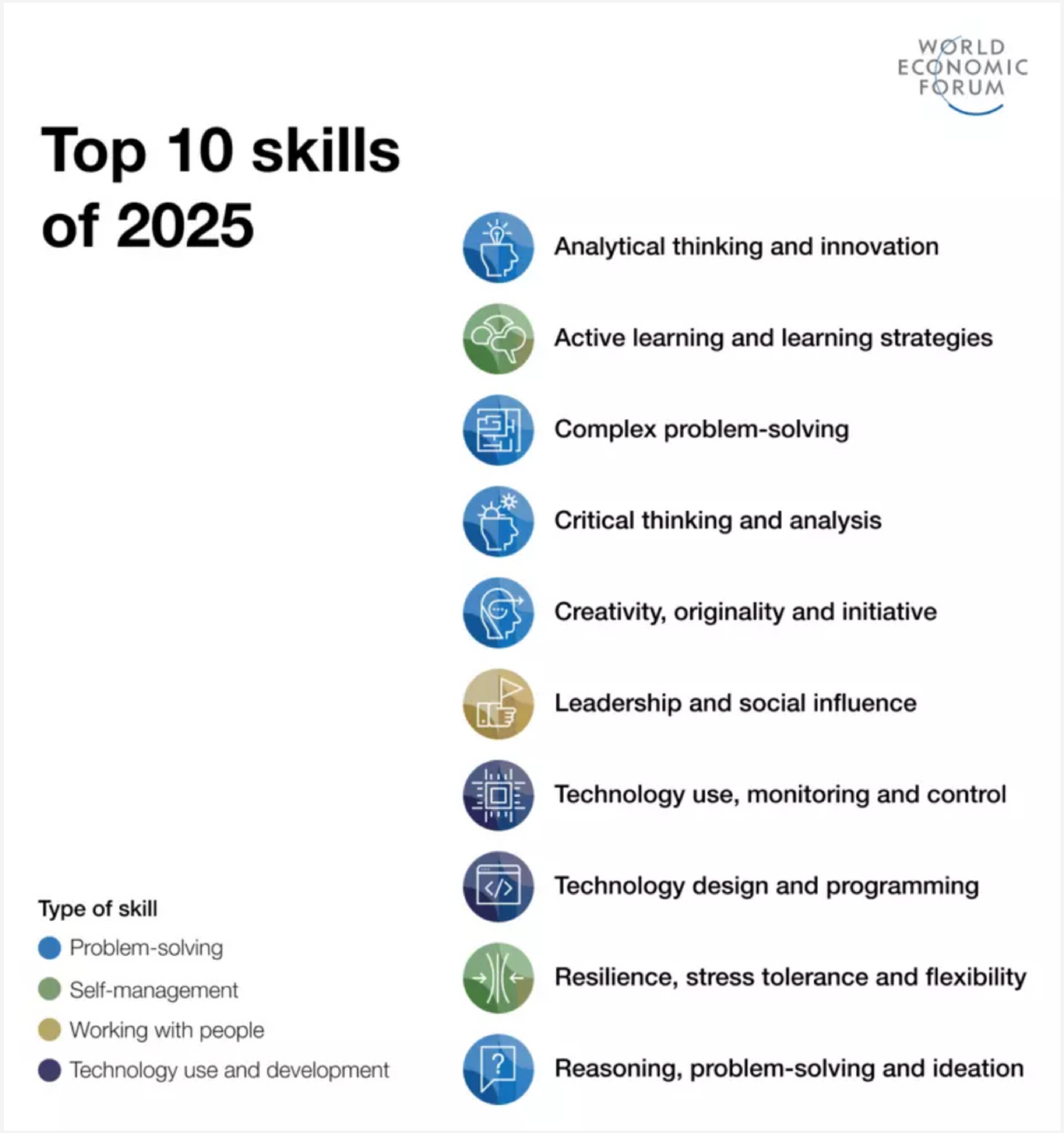
solve complex problems like a computer

“**Everybody** in the country should learn how to **program a computer** … because it teaches you **how to think**.”

Steve Jobs, co-founder of Apple

3

best entry point into
the tech world



WORLD ECONOMIC FORUM

Top 10 skills of 2025

- Analytical thinking and innovation
- Active learning and learning strategies
- Complex problem-solving
- Critical thinking and analysis
- Creativity, originality and initiative
- Leadership and social influence
- Technology use, monitoring and control
- Technology design and programming
- Resilience, stress tolerance and flexibility
- Reasoning, problem-solving and ideation

Type of skill

- ● Problem-solving
- ● Self-management
- ● Working with people
- ● Technology use and development

WORLD ECONOMIC FORUM

Top 10 skills of 2025

→ Analytical thinking and innovation

Active learning and learning strategies

→ Complex problem-solving

→ Critical thinking and analysis

→ Creativity, originality and initiative

Leadership and social influence

→ Technology use, monitoring and control

→ Technology design and programming

Resilience, stress tolerance and flexibility

→ Reasoning, problem-solving and ideation

Type of skill

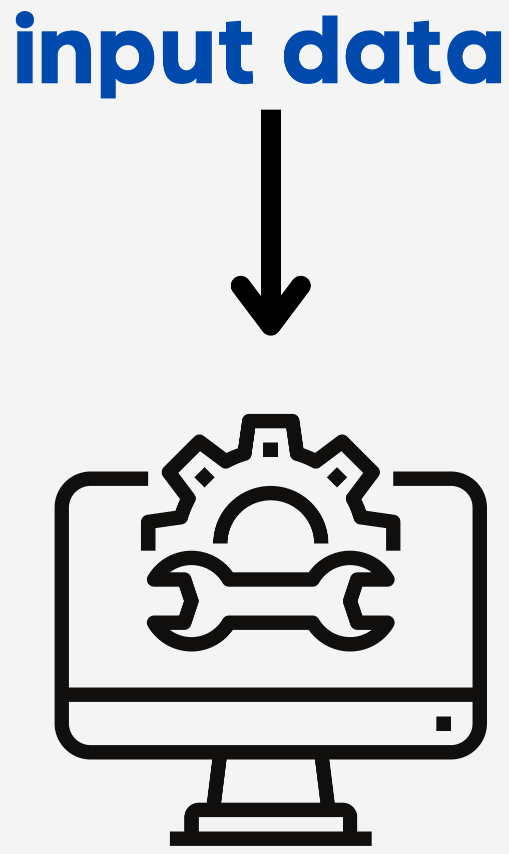
● Problem-solving

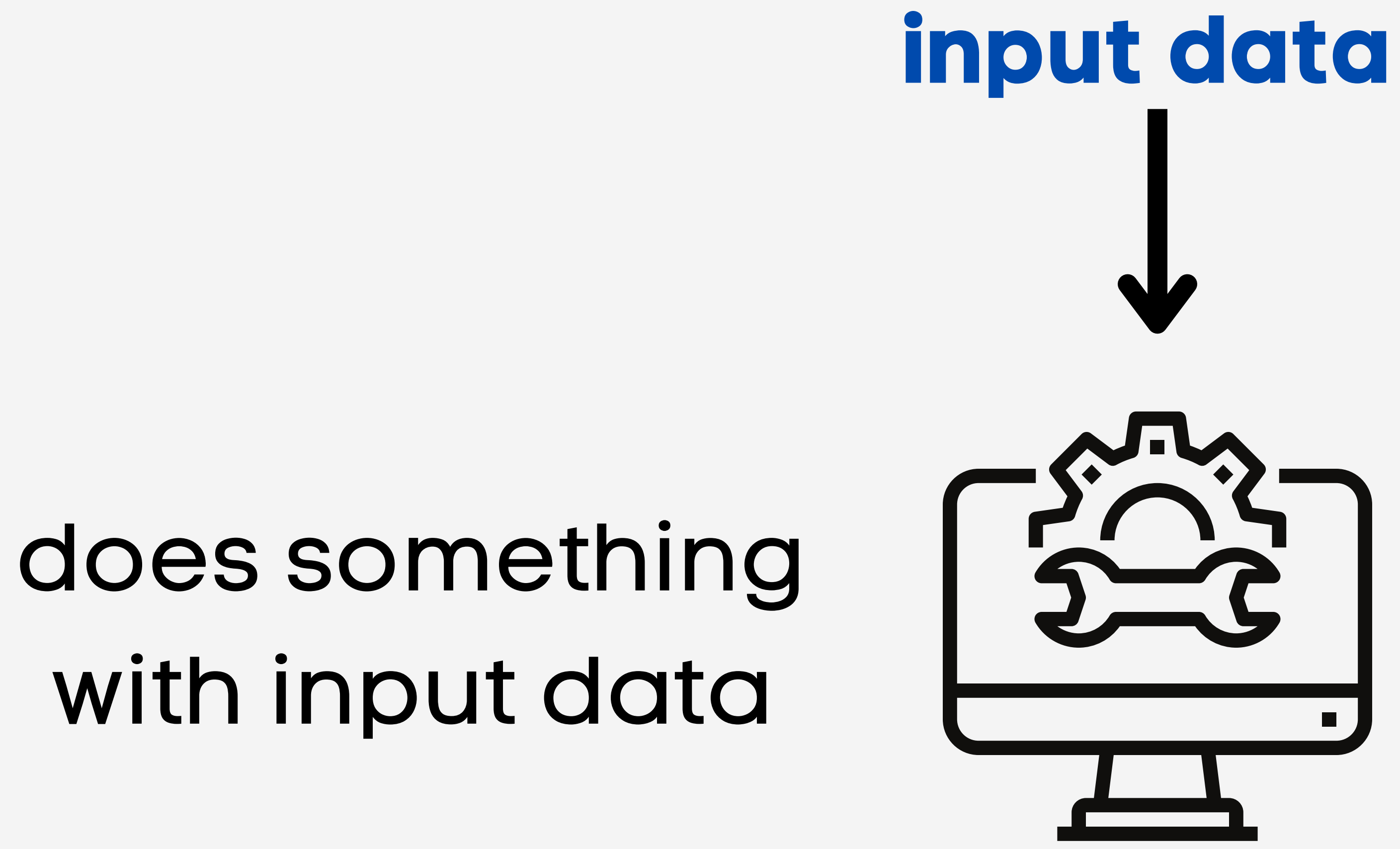
● Self-management

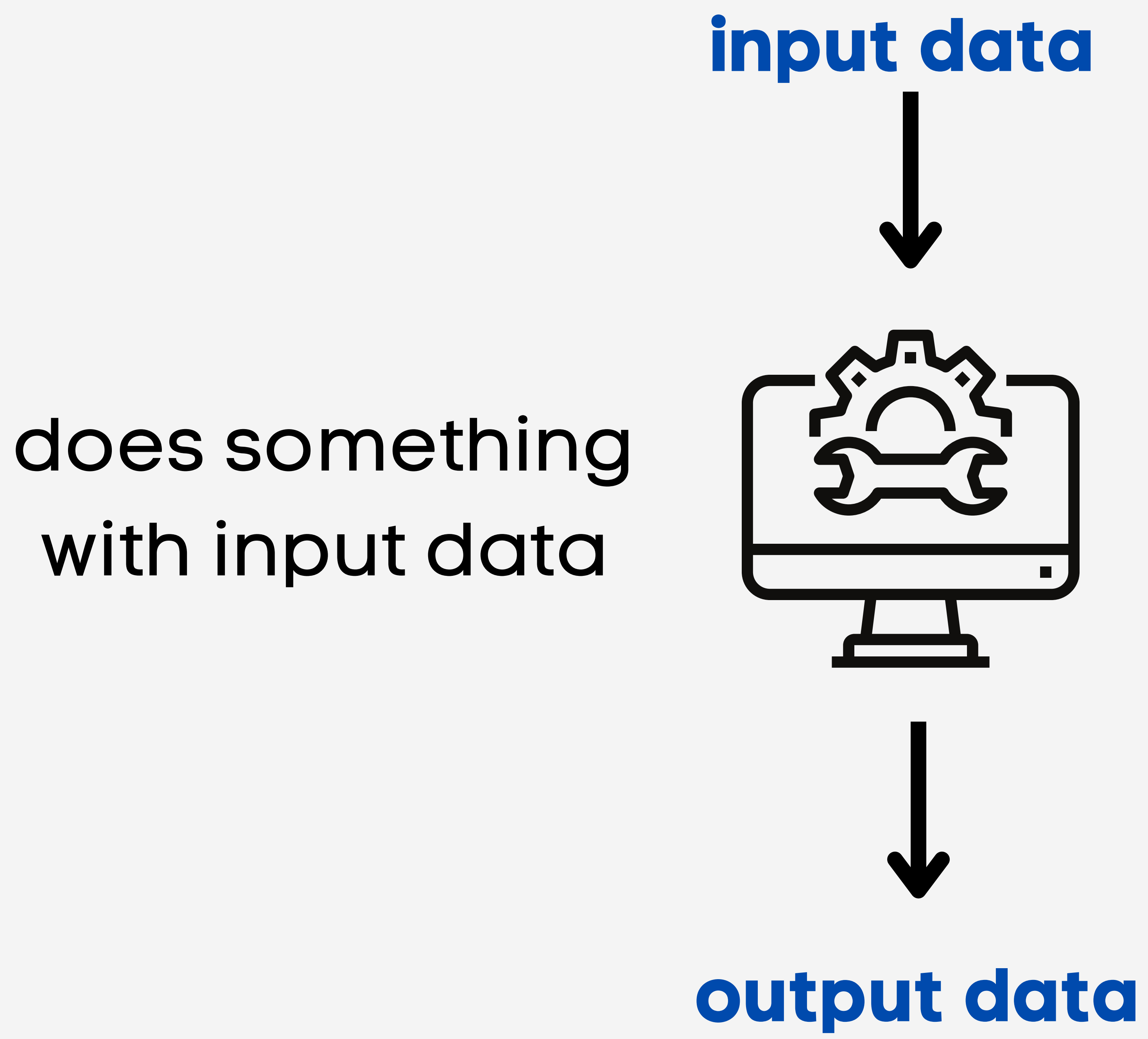
● Working with people

● Technology use and development

what does a computer program do?


input data


input data
does something
with input data


input data
does something
with input data
output data

let's take a look at **everyday examples** calculator

input data

mathematical expression

input data
mathematical expression

↓

calculator

does the math
transforms input data

input data
mathematical expression

↓

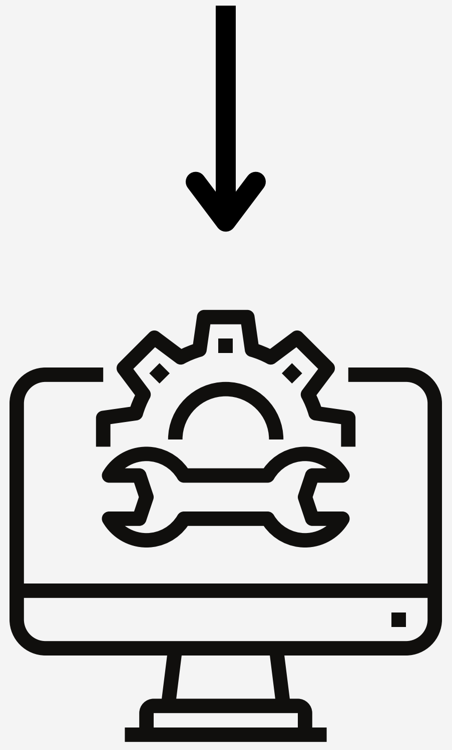
calculator
does the math
transforms input data

↓

output data
result of the computation

phone facial recognition

input data
image of your face



input data
image of your face

↓

algorithm transforms
the image
transforms input data

input data
image of your face

$\downarrow$

algorithm transforms
the image
transforms input data

$\downarrow$

output data
Yes you can login
No you can't

as programmers, your role is to write **what the program does**

DATA TYPES & VARIABLES

what **types of data** can we manipulate with Python?

strings are for literal text

```
"this is a string"
```

a string can be a single letter, a full sentence, or even an entire textbook

```
"I am 29 years old"
```

```
"5684297"
```

"Horatio says 'tis but our fantasy,
And will not let belief take hold of
him touching this dreaded sight, twice
seen of us. Therefore I have entreated
him along ..."

qutoes must be
the same

```
"5684297'
```

integers are for whole numbers

15

```
15
```

```
-200
```

```
0
```

floats are for decimal numbers

3.14

3.14

-156.52628

0.000

booleans are for True or False statements

True

False

booleans are used to evaluate statements

$1 > 5$

```
1 > 5
```

Is 1 greater than 5?

```
1 > 5
```

```
False
```

```
36 == 36
```

Is 36 equal to 36?

```
36 == 36
```

```
True
```

Four basic data types

```
"Hello world!"
```

```
2986
```

```
259.146
```

```
True
```

what can we do with those data?

we can perform math computations on **numerical data**

26 + 58

84

```
26.5 / 3
```

```
8.83333
```

in Python, **strings** can also be added together

```
"Hello " + "world!"
```

```
"Hello " + "world!"
```

```
"Hello world!"
```

```
"Hello " + "world!"
```

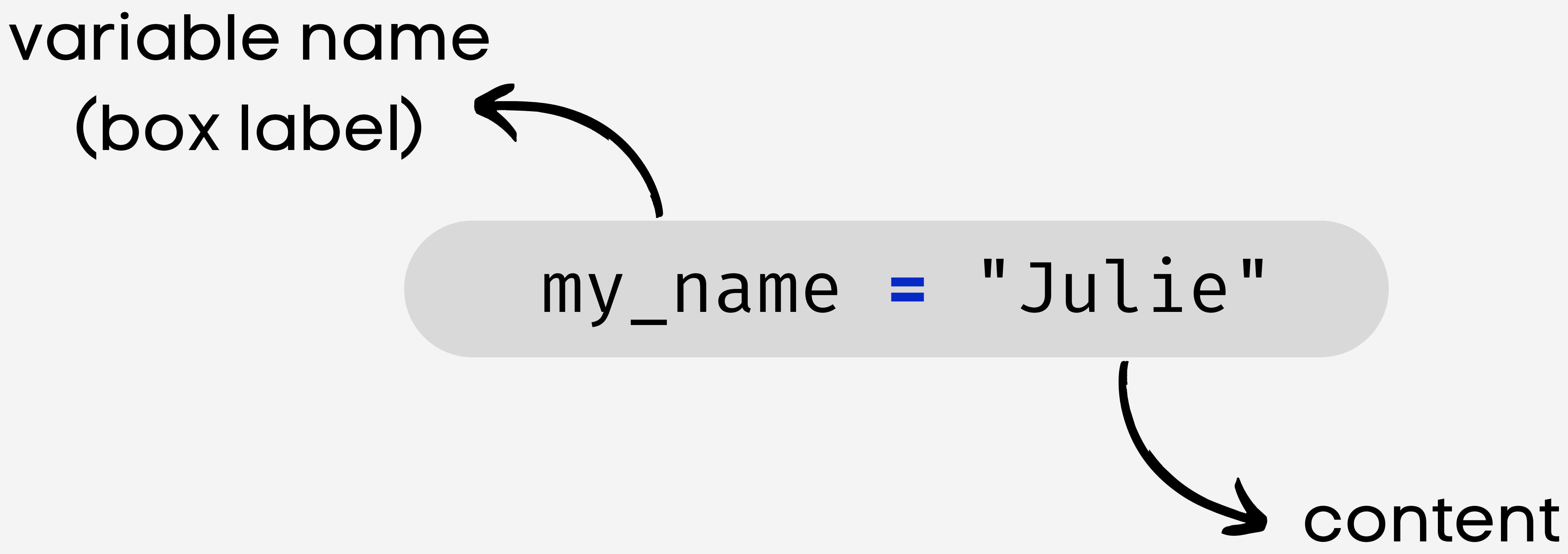
```
"Hello world!"
```

try it yourself in
the playground

all types of data can be stored in memory using
variables

variables are just like **boxes** in your computer memory just like boxes, variables have a **label** and a **content** let's say I want to store my name **"Julie"** as a string inside a variable

```
my_name = "Julie"
```



```
my_name = "Julie"
```

```
my_age = 29
```

```
x = 29
```

```
variable_1 = 29
```

```
my_age = 29
```

```
x = 29
```

```
variable_1 = 29
```

always give
descriptive names

```
my_height_at_15_yo = 170
```

```
myheightat15yo = 170
```

easier to read

```
my_height_at_15_yo = 170
```

```
myheightat15yo = 170
```

now that my data has been stored in variables, I can **access** and **use** it

```
my_name = "Julie"
```

```
my_age = 29
```

```
"My name is " + my_name
```

```
"My name is " + my_name
```

try it yourself in the playground

```
"My name is " + my_name
```

```
"My name is Julie"
```

```
my_name + " is" + my_age + " years old"
my_name + " is" + my_age + " years old"
```

```
TypeError: can only concatenate str (not "int") to str
```

make use of **formatted** strings!

```
f"{my_name} is {my_age} years old"
```

```
f"{my_name} is {my_age} years old"
```

```
Julie is 29 years old
```

you can also **update** the content of your variables

```
my_age = 29
```

```
f"I am now {my_age} years old"
```

```python
my_age = 29
```

```python
f"I am now {my_age} years old"
```

```
I am now 29 years old
```

```
my_age = my_age + 2
```

```
f"In two years, I will be {my_age} years old"
```

```
my_age = my_age + 2
```

```
f"In two years, I will be {my_age} years old"
```

```
In two years I will be 31 years old
```

METHODS

methods are **transformations** that we apply to **data** methods are pieces of code that have **already been written** by someone else

Python is so powerful because of its **community** that everyday develops **new methods** for people like you and me to use there are methods for almost **EVERYTHING**

```
my_name = "Julie"
my_name = "Julie"
```

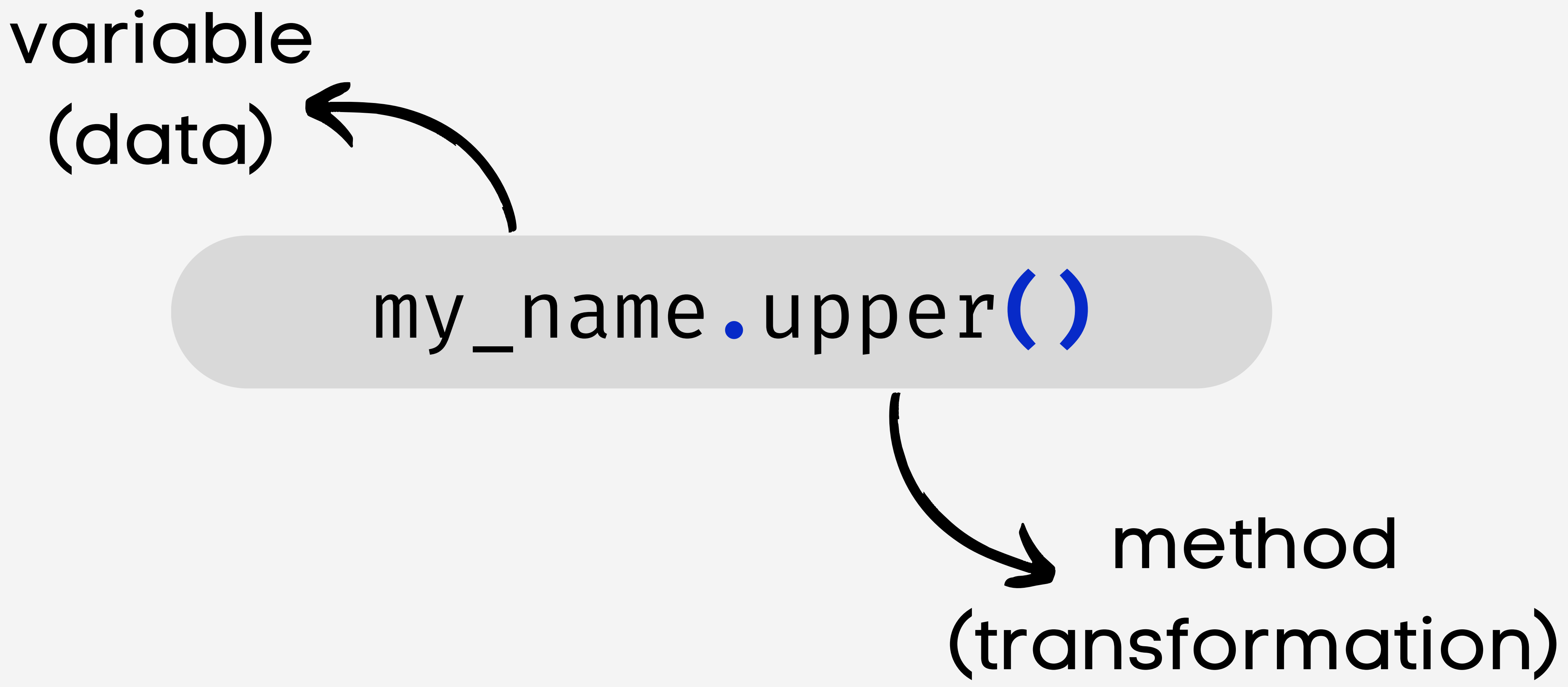
how can I write "Julie" in all uppercase?

Option 1: update manually

```
my_name = "JULIE"
```

Option 2: use a method!

```
my_name.upper()
```

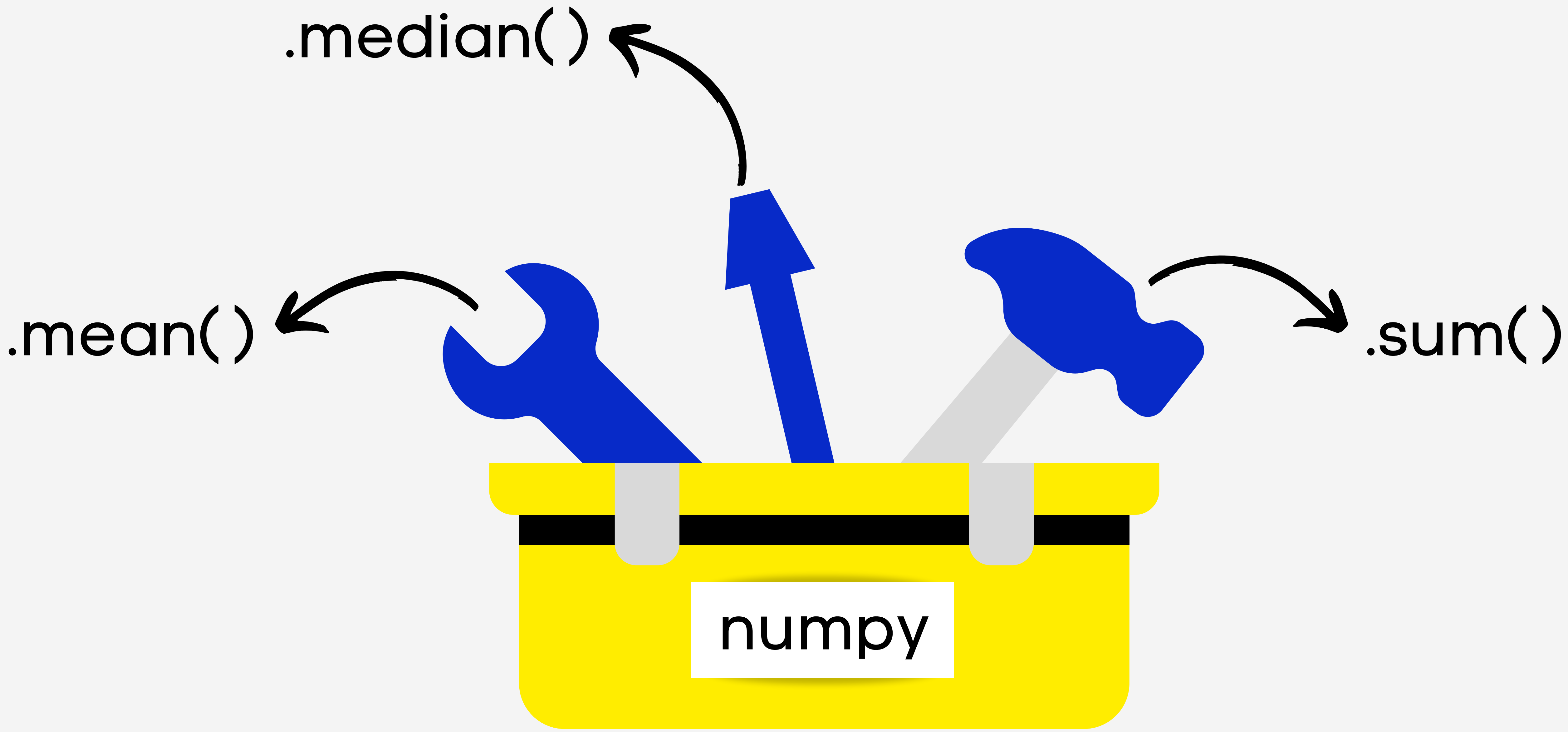


```
my_name.upper()
my_name.upper()
```

```
"JULIE"
```

Python already comes with a range of **built-in** ready-to-use methods

methods are often grouped in **libraries** for example, the **NumPy library** has thousands of **methods** for scientific computation


.median( )
.mean( )
.sum( )
numpy

other methods need to be **imported** inside your workspace

```
import numpy
```

```python
from numpy import mean
from numpy import mean
```

from the NumPy library, import the method mean

```python
from numpy import mean
```

from the toolbox NumPy, get the mean( ) tool

ITERABLE DATA TYPES

in many situations, we want to store **more than one** piece of data inside a **single variable**

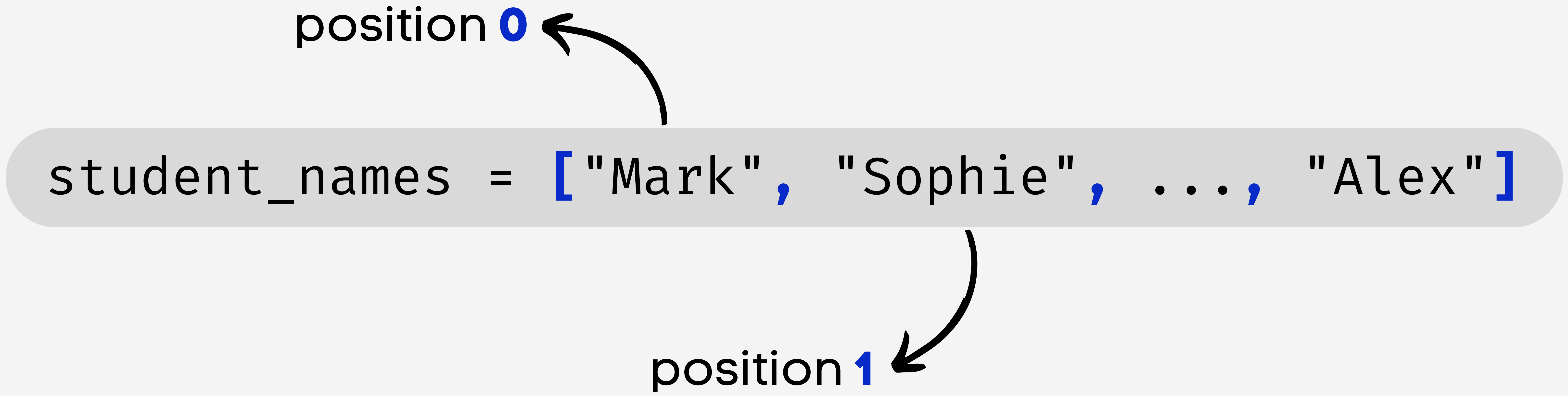
```
student_name_1 = "Mark"
student_name_2 = "Sophie"
...
student_name_20 = "Alex"
```

I can store all my students' names in a **list**

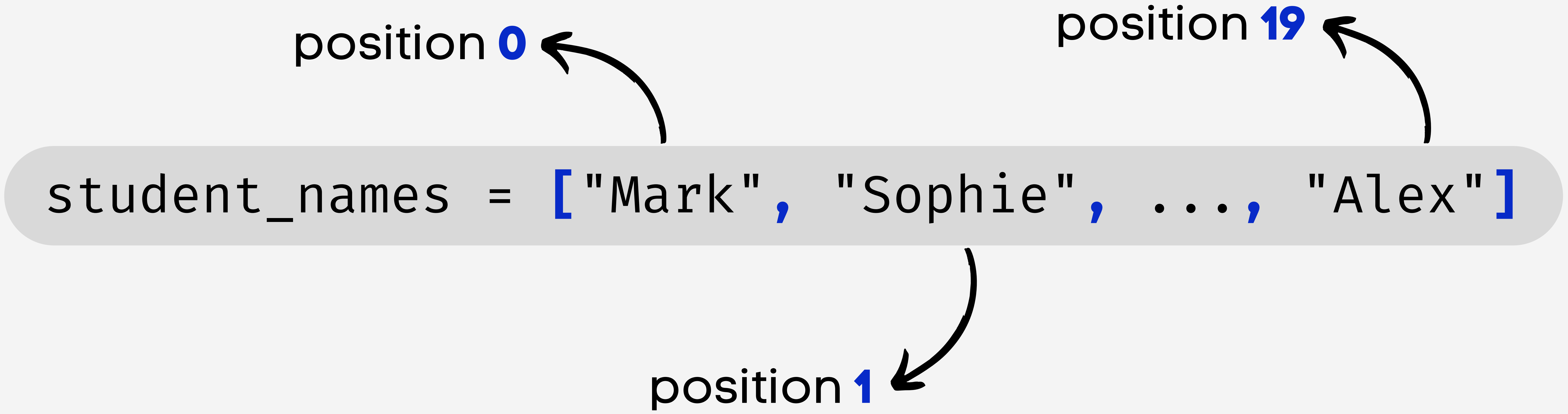
```
student_names = ["Mark", "Sophie", ..., "Alex"]
```

a list is a **sequence** of elements, each element has a position in the list

```python
student_names = ["Mark", "Sophie", ..., "Alex"]
```



```python
student_names = ["Mark", "Sophie", ..., "Alex"]
```



```
student_names = ["Mark", "Sophie", ..., "Alex"]
```

how do I **access** my students' names?

```
student_names[0]
```

```
student_names[0]
```

```
"Mark"
```

```
"The name of my first student is " + student_names[0]
```

```
"The name of my first student is Mark"
```

LOOPS

```
student_names = ["Mark", "Sophie", ..., "Alex"]
```

say I want to print out all my students' names

```
"The name of my first student is " + student_names[0]
"The name of my second student is " + student_names[1]
...
"The name of my 20th student is " + student_names[19]
```

I can use a **for loop** to iterate on my list of student names

```python
for student_name in student_names:
    print(student_name)
```

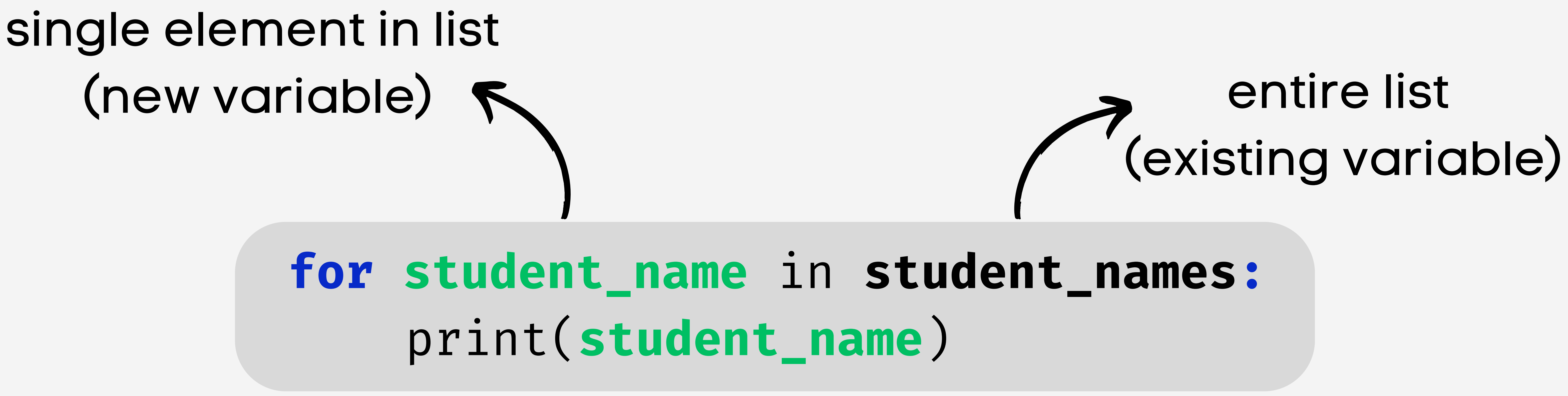
```python
for student_name in student_names:
    print(student_name)
```

for each student name in my list of names, print out the student name

single element in list
(new variable)

```
for student_name in student_names:
    print(student_name)
```

for each student name in my list of names, print out the student name



```
for student_name in student_names:
    print(student_name)
```

for each student name in my list of names, print out the student name

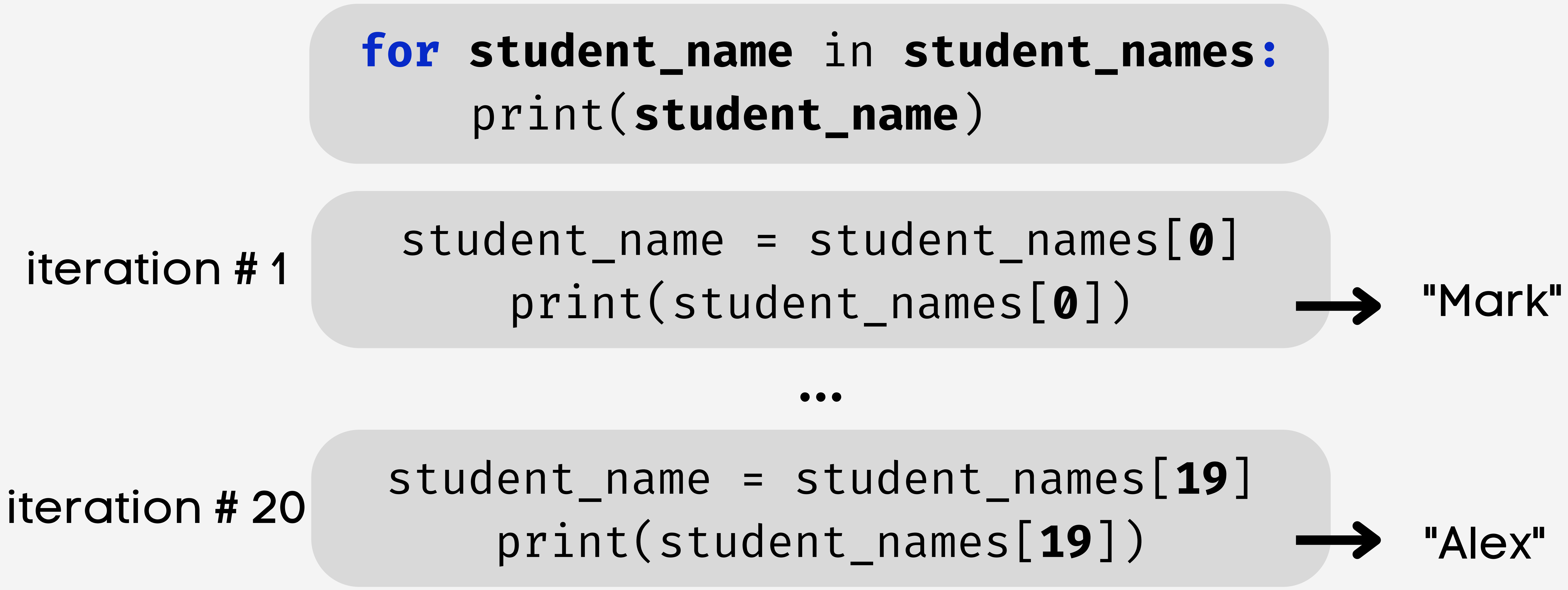
```
for student_name in student_names:
    print(student_name)
```

iteration # 1

```
student_name = student_names[0]
print(student_names[0])
```

$\rightarrow$ "Mark"

...



```python
for student_name in student_names:
    print(student_name)
```

iteration # 1

```python
student_name = student_names[0]
print(student_names[0])
```

→ "Mark"

...

iteration # 20

```python
student_name = student_names[19]
print(student_names[19])
```

→ "Alex"

```python
for student_name in student_names:
    print(student_name)
```

```
"Mark"
"Sophie"
...
"Alex"
```

say I want to spell out all my students' names in upper case

```python
for student_name in student_names:
    print(student_name.upper())
```

```python
for student_name in student_names:
    print(student_name.upper())
```

for each student name in my list of
names, spell student name in upper case

```python
for student_name in student_names:
    print(student_name.upper())
```

```python
for student_name in student_names:
    print(student_name.upper())
```

```
"MARK"
"SOPHIE"
...
"ALEX"
```

```python
for element in list_of_elements:
    # do something with element
```

```python
for element in list_of_elements:
    # do something with element
```

for each element in my list of elements,
do something with that element

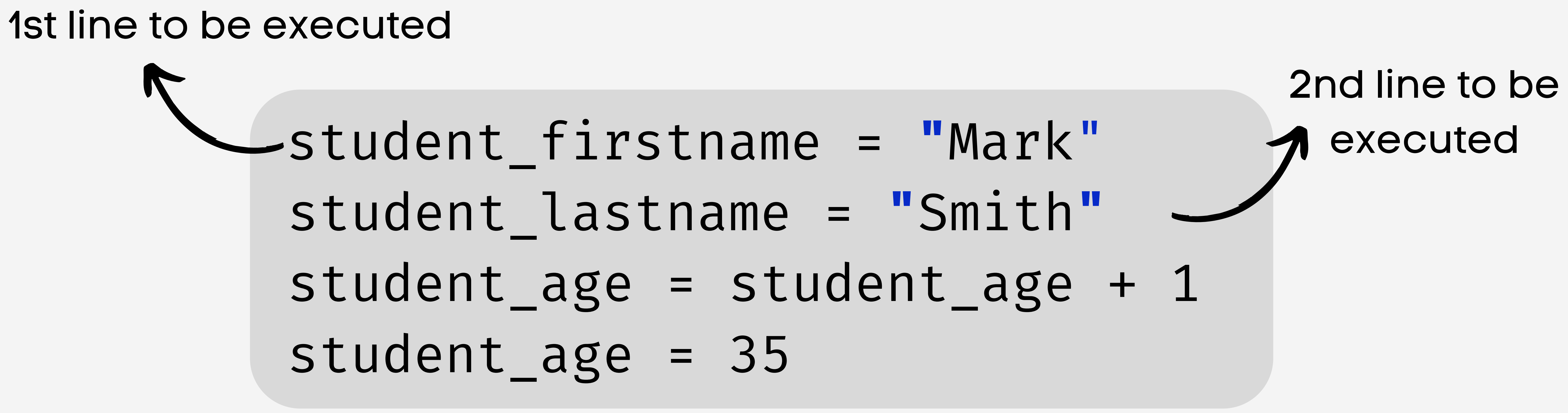
CONTROL FLOW

computers read and execute code **line by line**

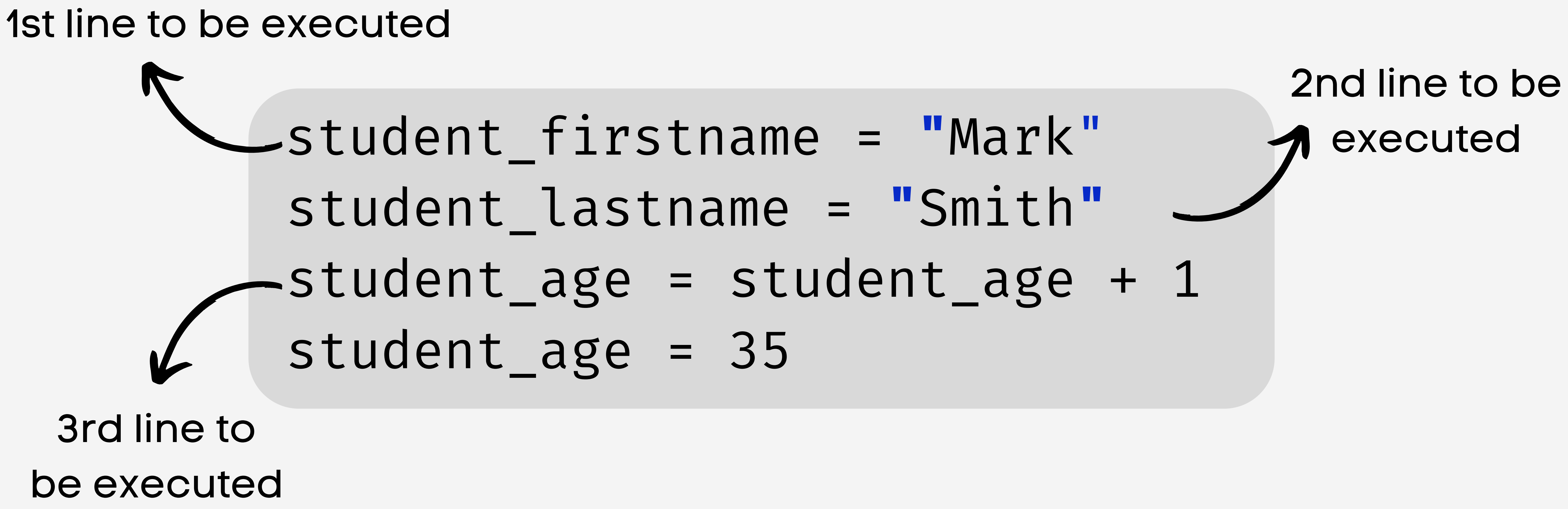
```
student_firstname = "Mark"
student_lastname = "Smith"
student_age = student_age + 1
student_age = 35
```

1st line to be executed

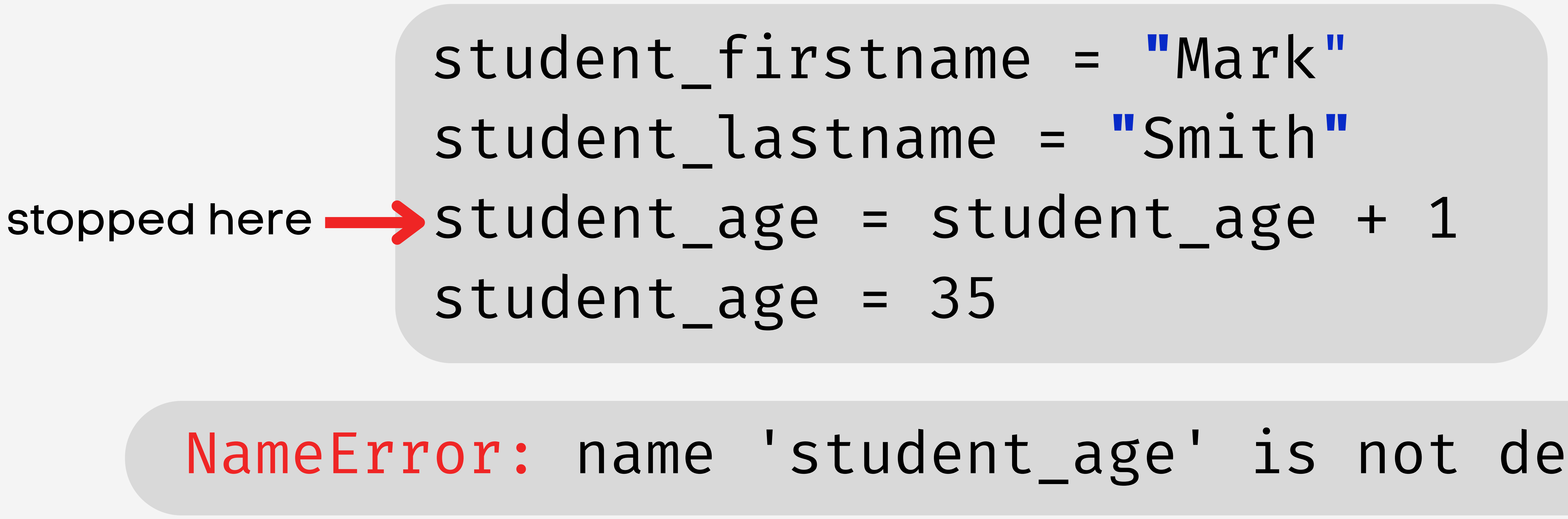
```
student_firstname = "Mark"
student_lastname = "Smith"
student_age = student_age + 1
student_age = 35
```



```
student_firstname = "Mark"
student_lastname = "Smith"
student_age = student_age + 1
student_age = 35
```

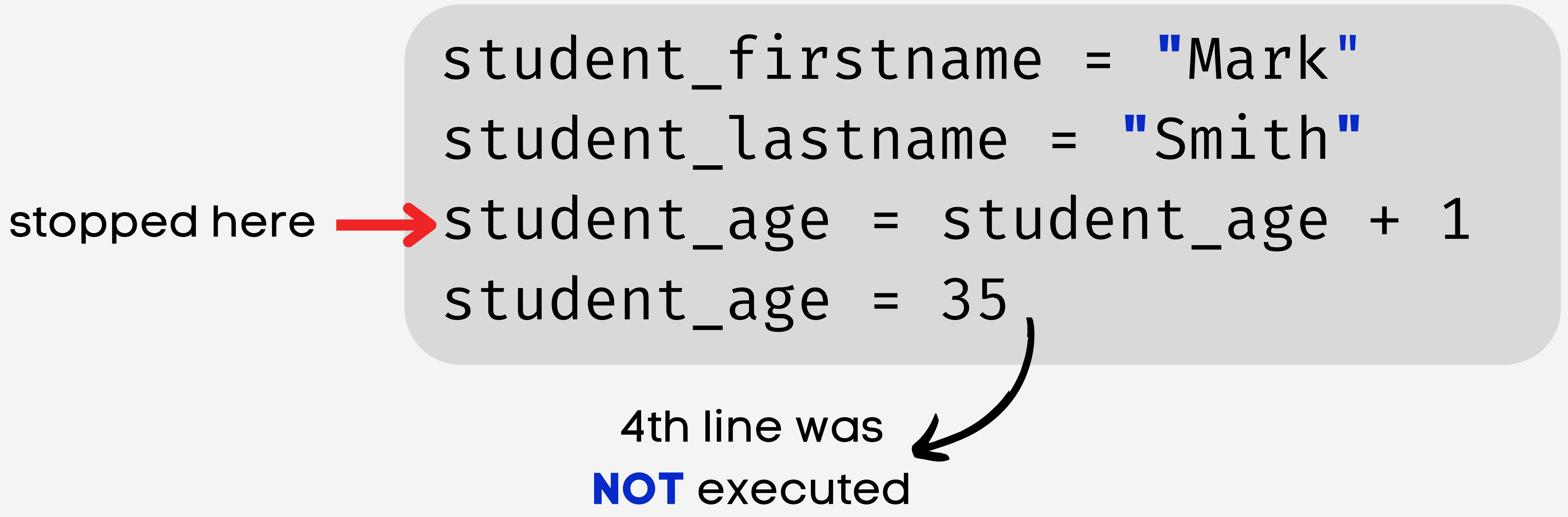


```
student_firstname = "Mark"
student_lastname = "Smith"
student_age = student_age + 1
student_age = 35
```



```python
student_firstname = "Mark"
student_lastname = "Smith"
student_age = student_age + 1
student_age = 35
```

```
NameError: name 'student_age' is not defined
```



```
student_firstname = "Mark"
student_lastname = "Smith"
student_age = student_age + 1
student_age = 35
```

```
student_firstname = "Mark"
student_lastname = "Smith"
student_age = student_age + 1
student_age = 35
```

```
student_firstname = "Mark"
student_lastname = "Smith"
student_age = 35
student_age = student_age + 1
```

student_name is now
defined before it is modified

in some situations, we might want to disrupt this sequential execution

Let's take student Mark as an example

Mark is 16 years old

```
mark_age = 16
```

let's say I want a program that tells me whether Mark is allowed to buy alcohol or not

```
mark_age = 16
print("Mark can buy alcohol")
```

```
mark_age = 16
if Mark is old enough:
    print("Mark can buy alcohol")
```

```python
mark_age = 16
if mark_age >= 19:
    print("Mark can buy alcohol")
```

1

```python
mark_age = 16
if mark_age >= 19:
    print("Mark can buy alcohol")
```

```
1 mark_age = 16
2 if mark_age >= 19:
      print("Mark can buy alcohol")
```

```
1 mark_age = 16
2 if mark_age >= 19:
      print("Mark can buy alcohol")
```

won't be executed

I also want my program to tell me if Mark is allowed to drive

```
mark_age = 16
if mark_age >= 19:
  print("Mark can drink an drive, not at the same time!")
elif Mark is old enough to drive:
  print("Mark can drive")
```

```python
mark_age = 16
if mark_age >= 19:
  print("Mark can drink an drive")
elif mark_age >= 16:
  print("Mark can drive")
```

Finally I want my program to tell me if Mark is neither allowed to drink alcohol or drive

```python
mark_age = 16
if mark_age >= 19:
  print("Mark can drink and drive")
elif mark_age >= 16:
  print("Mark can drive")
else:
  print("Mark cannot drink or drive")
```

let's look at the execution flow when Mark is 15

```python
mark_age = 15
if mark_age >= 19:
  print("Mark can drink and drive")
elif mark_age >= 16:
  print("Mark can drive")
else:
  print("Mark cannot drink or drive")
```

```
1 mark_age = 15
2 if mark_age >= 19:
    print("Mark can drink and drive")
3 elif mark_age >= 16:
    print("Mark can drive")
4 else:
5   print("Mark cannot drink or drive")
```

`"Mark cannot drink or drive"`

how about when Mark is 17?

```
1 mark_age = 17
2 if mark_age >= 19:
    print("Mark can drink and drive")
3 elif mark_age >= 16:
4   print("Mark can drive")
  else:
    print("Mark cannot drink or drive")
```

```
"Mark can drive"
```

and what happens if Mark is 21?

```
1 mark_age = 21
2 if mark_age >= 19:
3     print("Mark can drink and drive")
  elif mark_age >= 16:
      print("Mark can drive")
  else:
      print("Mark cannot drink or drive")
```

```
"Mark can drink and drive"
```

FUNCTIONS

Make it a function!

Remember **methods**? Those pieces of code that have been already written by someone else…

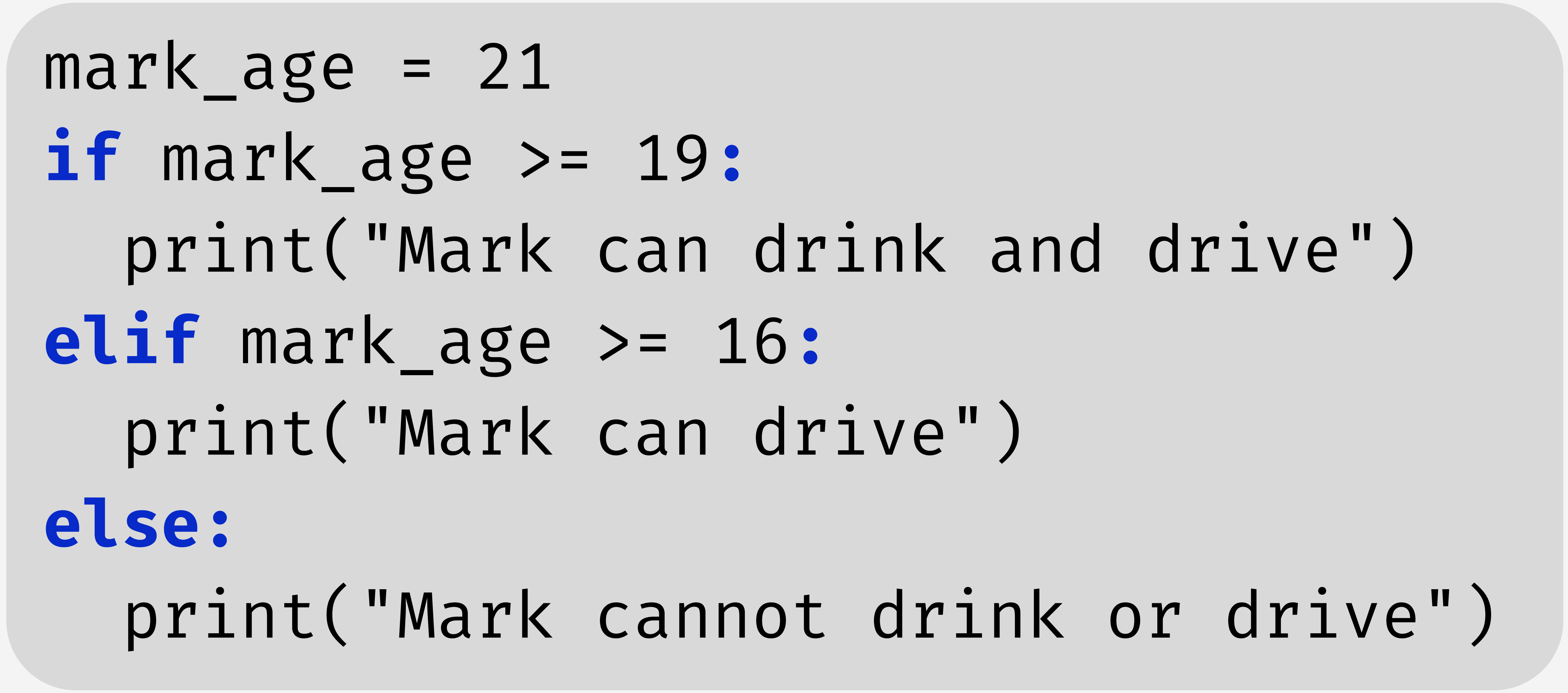
well this time, "someone else" is **you**!

functions are small pieces of code that you can

reuse

let's take Mark as an example

no matter his age, I want my program to tell me
whether Mark is allowed to drink and/or drive.



```python
mark_age = 21
if mark_age >= 19:
  print("Mark can drink and drive")
elif mark_age >= 16:
  print("Mark can drive")
else:
  print("Mark cannot drink or drive")
```

```python
def what_can_mark_do():
  mark_age = 15
  if mark_age >= 19:
    print("Mark can drink and drive")
  elif student_age >= 16:
    print("Mark can drive")
  else:
    print("Mark cannot drink or drive")
```

now that my function is defined, I can **call** it `what_can_mark_do()`

```
what_can_mark_do()
```

```
"Mark cannot drink or drive"
```

```python
def what_can_mark_do():
  mark_age = 15
  if mark_age >= 19:
    print("Mark can drink and drive")
  elif student_age >= 16:
    print("Mark can drive")
  else:
    print("Mark cannot drink or drive")
```

Mark’s age always set to
15 yo, not very flexible...

```
def what_can_mark_do():
  mark_age = 15
  if mark_age >= 19:
    print("Mark can drink and drive")
  elif student_age >= 16:
    print("Mark can drive")
  else:
    print("Mark cannot drink or drive")
```

```python
def what_can_mark_do(mark_age):
  if mark_age >= 19:
    print("Mark can drink and drive")
  elif student_age >= 16:
    print("Mark can drive")
  else:
    print("Mark cannot drink or drive")
```

mark_age is now an "argument"

```
def what_can_mark_do(mark_age):
  if mark_age >= 19:
    print("Mark can drink and drive")
  elif student_age >= 16:
    print("Mark can drive")
  else:
    print("Mark cannot drink or drive")
```

```
what_can_mark_do(mark_age = 17)
what_can_mark_do(mark_age = 17)
```

```
"Mark can drive"
```

```
what_can_mark_do(mark_age = 21)
```

```
"Mark can drink and drive"
```

mark_age
input data

what_can_mark_do( )
tests conditions on
mark_age

prints a string
"Mark can drive"

```python
def mark_age_in_2025(mark_age):
    mark_age = mark_age + 2
```

```python
def mark_age_in_2025(mark_age):
    mark_age = mark_age + 2
    return mark_age
```

mark_age
input data

↓

what_age_in_2025( )
adds 2 years to
mark_age

↓

output data
updated mark_age

```
mark_age_in_2025(mark_age = 18)
```

```
mark_age_in_2025(mark_age = 18)
```

```
20
```

```
mark_age_in_2025(mark_age = 34)
```

```
36
```

```
older_mark_age = mark_age_in_2025(mark_age = 34)
```

```
older_mark_age
36
```

DEBUGGING & WORKFLOW

what to do if your code doesn’t work?

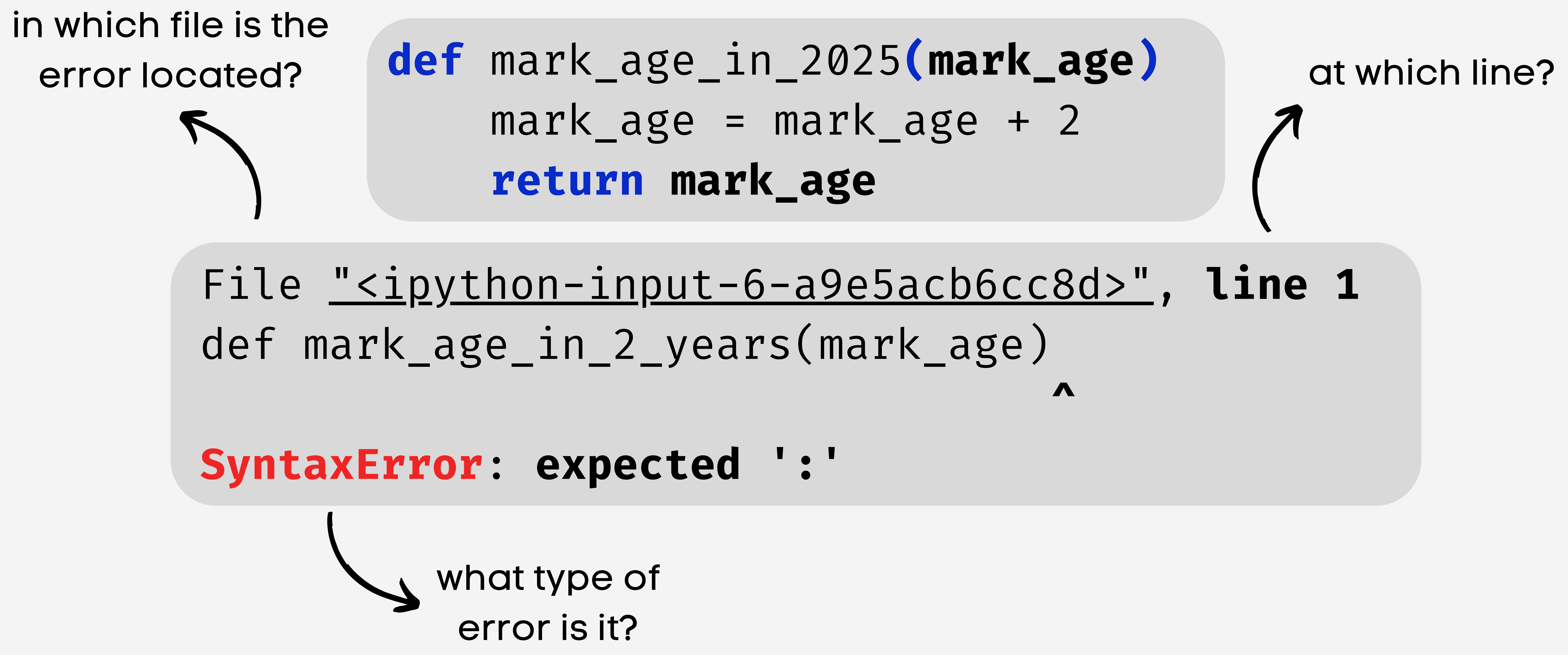
```
def mark_age_in_2025(mark_age)
    mark_age = mark_age + 2
    return mark_age
```

```python
def mark_age_in_2025(mark_age)
    mark_age = mark_age + 2
    return mark_age
```

```
File "<ipython-input-6-a9e5acb6cc8d>", line 1
def mark_age_in_2_years(mark_age)
                                 ^

SyntaxError: expected ':'
```

start by **reading** the error message!



```
def mark_age_in_2025(mark_age)
    mark_age = mark_age + 2
    return mark_age
```

```
File "<ipython-input-6-a9e5acb6cc8d>", line 1
def mark_age_in_2_years(mark_age)
                                 ^

SyntaxError: expected ':'
```

in which file is the error located?
at which line?
what type of error is it?

turns out a : is missing

```python
def mark_age_in_2025(mark_age):
    mark_age = mark_age + 2
    return mark_age
```

```python
def mark_age_in_2025(mark_age):
    mark_age = mark_age + 2
    print(f"In 2025, Mark will turn {student_age}")
```

```
def mark_age_in_2025(mark_age):
    mark_age = mark_age + 2
    print(f"In 2025, Mark will turn {student_age}")
```

```
File "<ipython-input-10-3ba94af1675b>", line 3
      2 mark_age = mark_age + 2
----> 3 print(f"Mark is {student_age} years old")
NameError: name 'student_age' is not defined
```

```
mark_age = 18
print("Mark is currently " + mark_age)
```

```python
mark_age = 18
print("Mark is currently " + mark_age)
```

```
File "<ipython-input-13-9333d0f22dae>", line 2
      1 mark_age = 18
----> 2 print("Mark is currently " + mark_age)
TypeError: can only concatenate str (not "int") to str
```

```
student_names = ["Mark, "Sophie", "Carol"]
print("The third student is " + student_names[3])
```

```
File "<ipython-input-15-3920c14cafba>" in line 2
      1 student_names = ["Mark", "Sophie", "Carol"]
----> 2 print("The third student is " + student_names[3])
IndexError: list index out of range
```

```
student_ages = [18, 22, 26]
student_ages[0].upper()
```

```python
student_ages = [18, 22, 26]
student_ages[0].upper()
```

```
File <ipython-input-16-bafa93cf7c99>", line 2
      1 student_ages = [18, 22, 26]
----> 2 student_ages[0].upper()
AttributeError: 'int' object has no attribute 'upper'
```

where & how to find **help**?

GOOGLE

GOOGLE

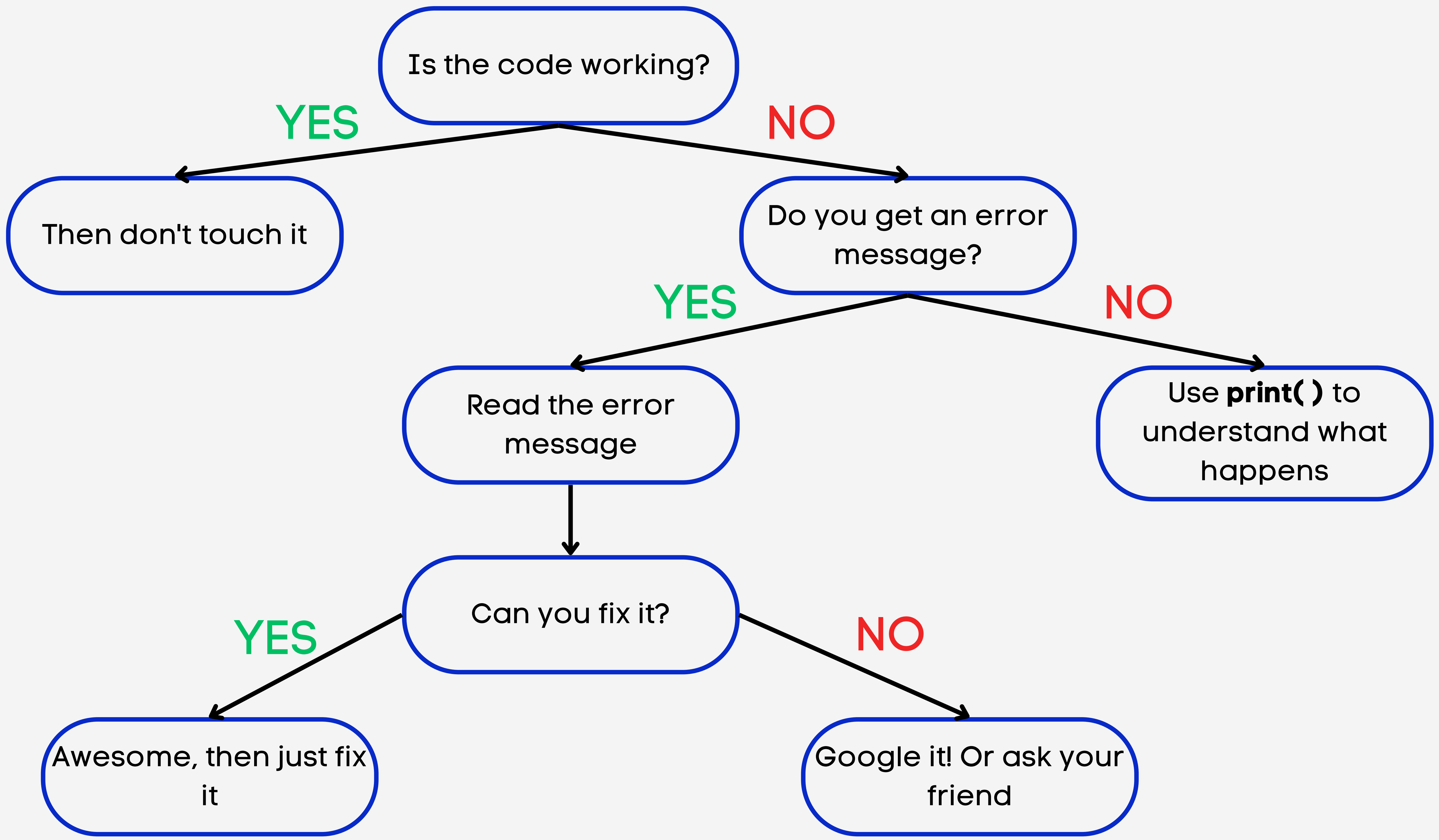
whatever it is, just google it!

STACKOVERFLOW

<u>W3 SCHOOLS</u>

CODING FRIENDS & COLLEAGUES

DEBUGGING WORKFLOW


Is the code working?
YES
NO
Then don't touch it
Do you get an error message?
YES
NO
Read the error message
Use **print( )** to understand what happens
Can you fix it?
YES
NO
Awesome, then just fix it
Google it! Or ask your friend

YOUR TURN TO PRACTICE!

check out the Python **cheatsheet** and try to solve the **coding challenge**!