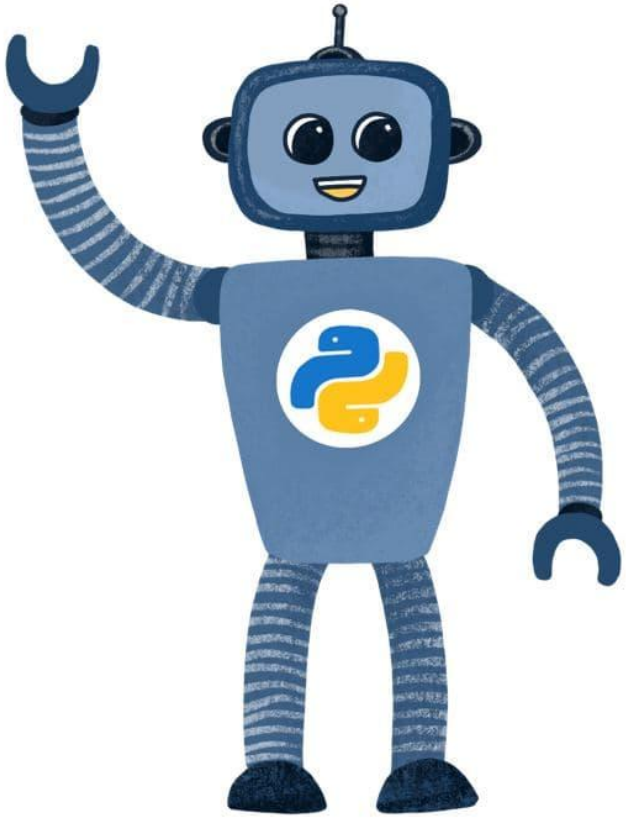
INTRODUCTION TO PYTHON

Aylin Kallmayer, Melvin Kallmayer, Leah Kumle

GRADE Workshop Frankfurt, May 11-12th 2021



This is Python...

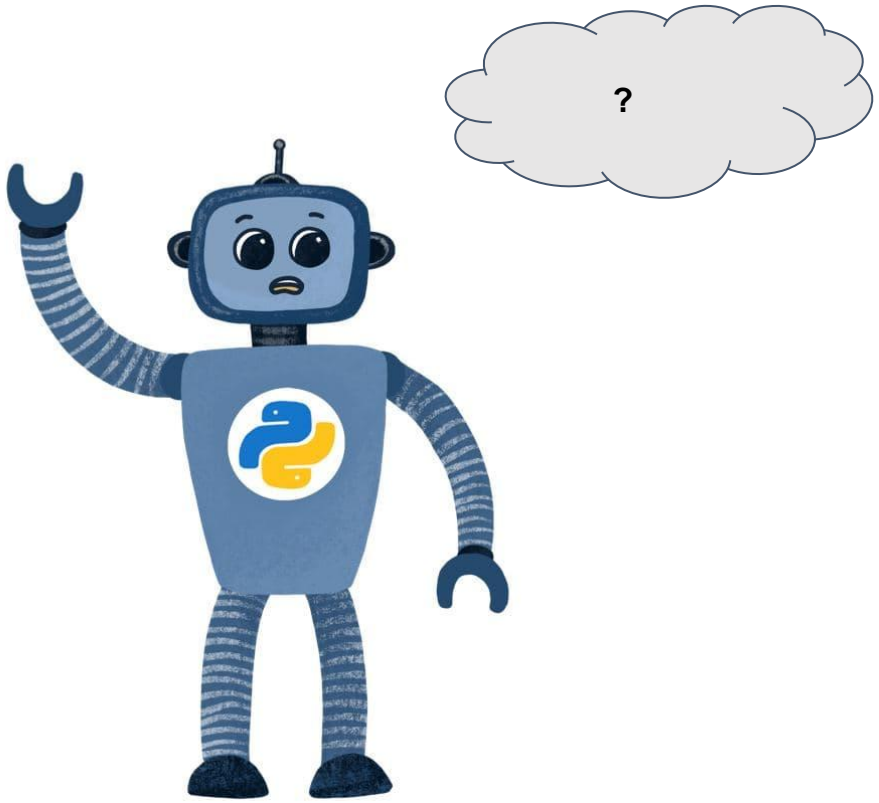
it is a very powerful and useful machine

… but we need to learn how to work together!

... which means learning its language!

PYTHON AS A FANCY CALCULATOR

“Python - What is 167263 times 65345?”



PYTHON AS A FANCY CALCULATOR

“Python - What is 167263 times 65345?”

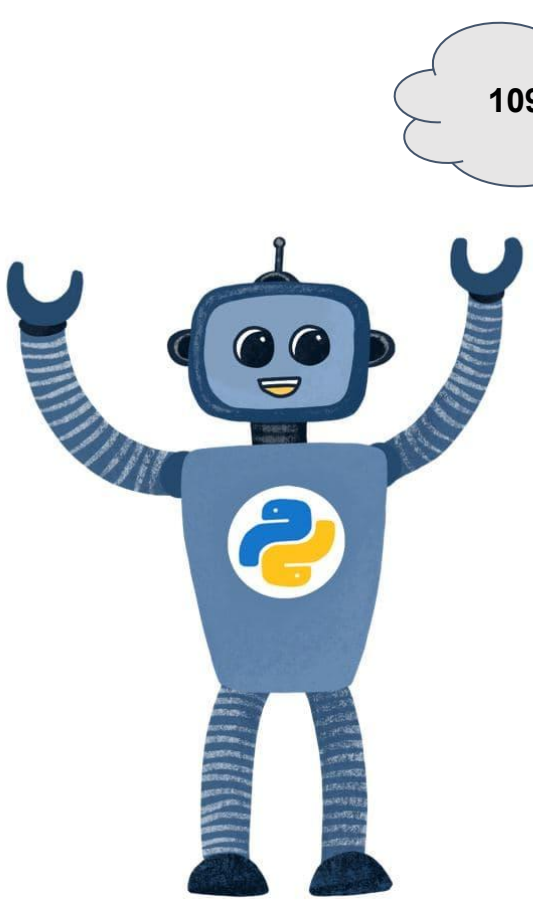
```
167263*65345
```

10929800735

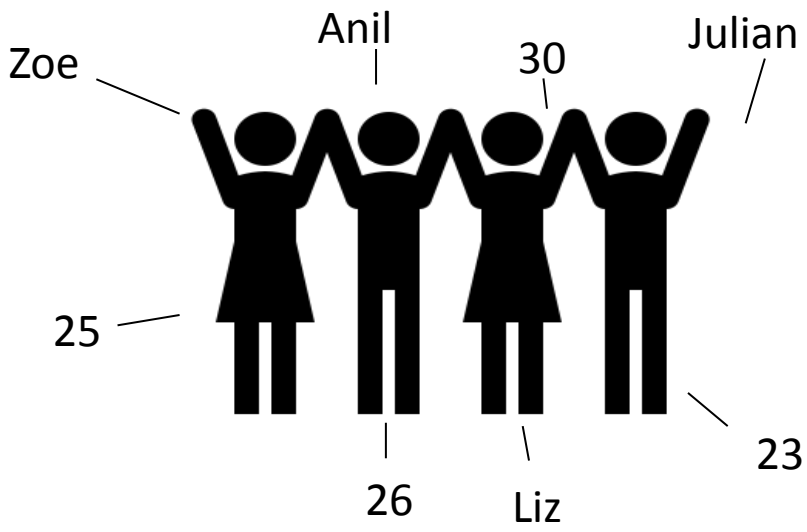
PYTHON AS A FANCY CALCULATOR

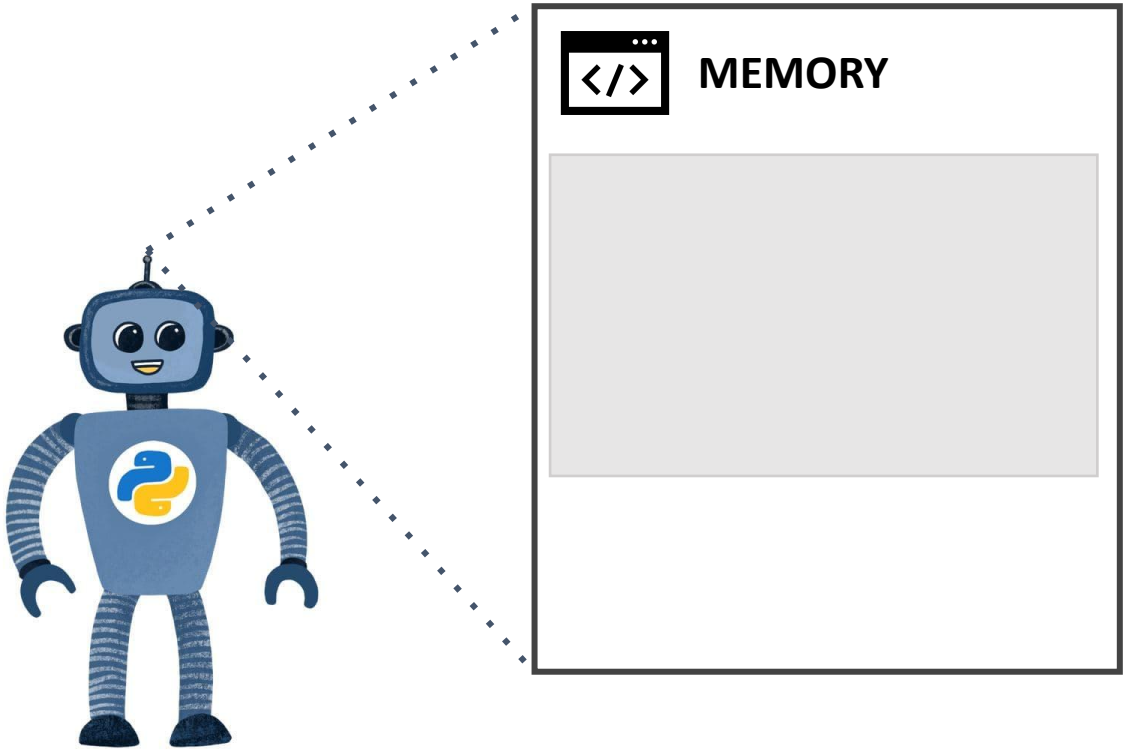
“Python - What is 167263 times 65345?”

+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponent



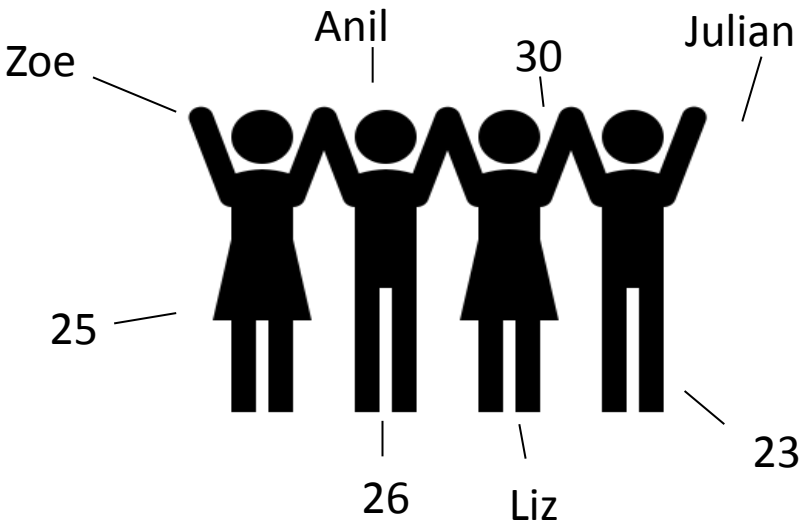
VARIABLES





VARIABLES

“Python - memorize the names and ages of all participants!”



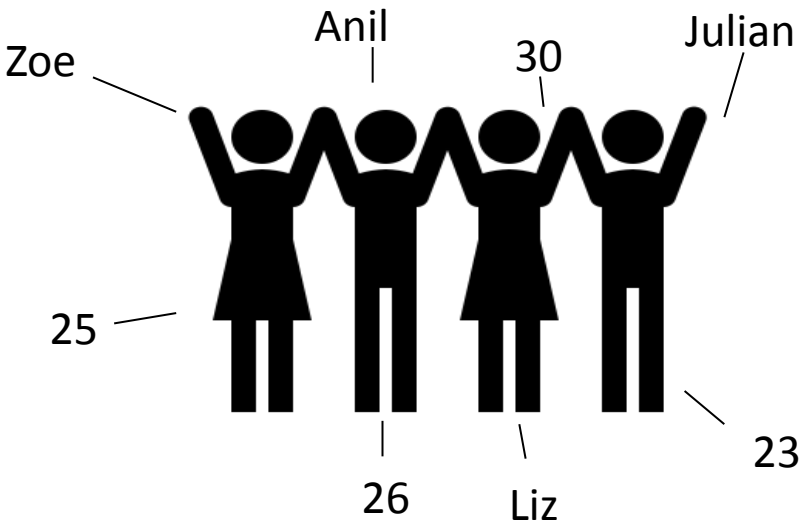
MEMORY

name1 = “Julian”
age_julian = 23

name2 = “Liz”
age_liz = 30

VARIABLES

“Python - How old are Julian and Liz combined?”



MEMORY

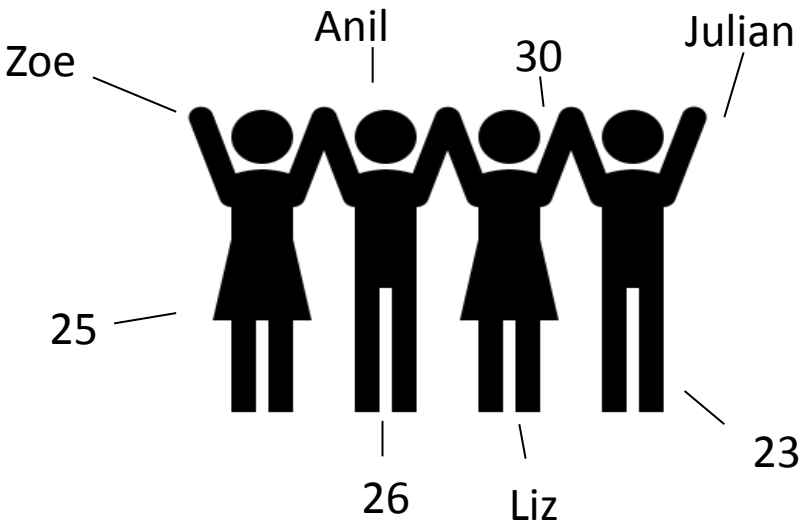
name1 = “Julian”
age_julian = 23

name2 = “Liz”
age_liz = 30

age_julian + age_liz
> 53

VARIABLES

“Python - How old are Julian and Liz combined?”



MEMORY

```
name1 = “Julian”
age_julian  = 23

name2 = “Liz”
age_liz = 30
```

```
age_julian + age_liz
> 53
```

Variables are used to store information
so that we can access and manipulate them later

TYPES

MEMORY

numbers	string	lists	dictionaries
age_julian = 23 age_liz = 30	name1 = “Julian” name2 = “Liz”		

TYPES

MEMORY

numbers	string	lists	dictionaries
age_julian = 23 age_liz = 30	name1 = “Julian” name2 = “Liz”		

age_julian + age_liz
> 53

name1 + name2
> “JulianLiz”

TYPES

MEMORY

numbers	string	lists	dictionaries
age_julian = 23 age_liz = 30	name1 = “Julian” name2 = “Liz”		

age_julian + name1
> **ERROR**

ERROR MESSAGES

age_julian + name1

> **ERROR**

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-13-afd3dfa0977f> in <module>
----> 1 age_julian + name1

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

ERROR MESSAGES

age_julian + name1

> **ERROR**

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-13-afd3dfa0977f> in <module>
----> 1 age_julian + name1

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

Error messages can be very useful!

ERROR MESSAGES

age_julian + name1

> **ERROR**

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-13-afd3dfa0977f> in <module>
----> 1 age_julian + name1

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

Error messages can be very useful!

ERROR MESSAGES

age_julian + name1

> **ERROR**

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-13-afd3dfa0977f> in <module>
----> 1 age_julian + name1

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

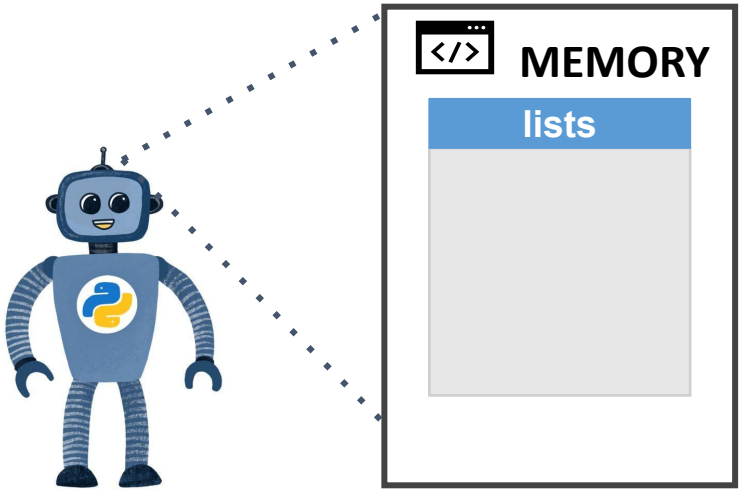
Error messages can be very useful!

ERROR MESSAGES

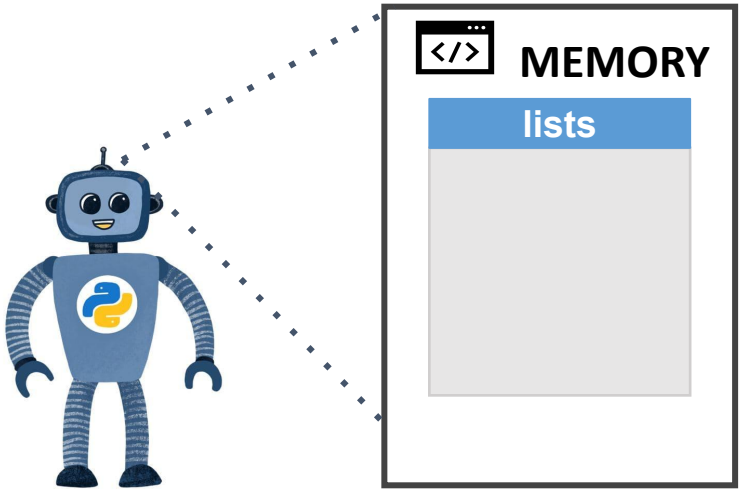
Other error messages:

- **TypeError:** Python can’t do a certain command with this variable type
- **NameError:** Python doesn’t know a variable with this name
- **SyntaxError:** Python doesn’t understand the grammar of our code

LISTS

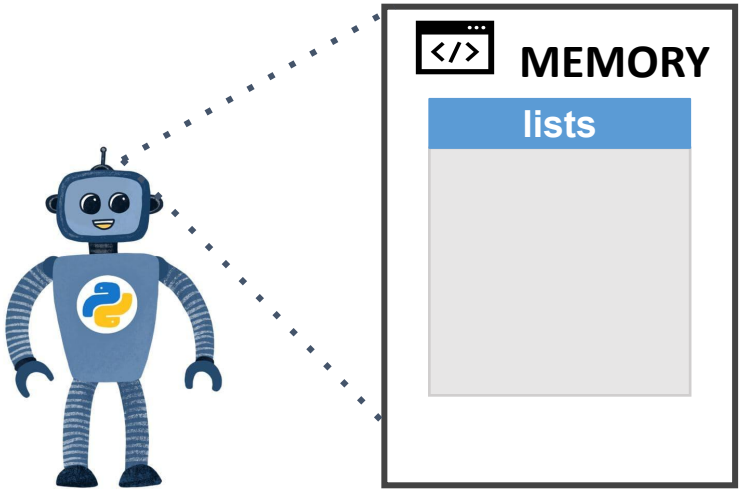


LISTS



pythons_list = [“Hello”, “my”, “name”, “is”, “Python”, “!”]

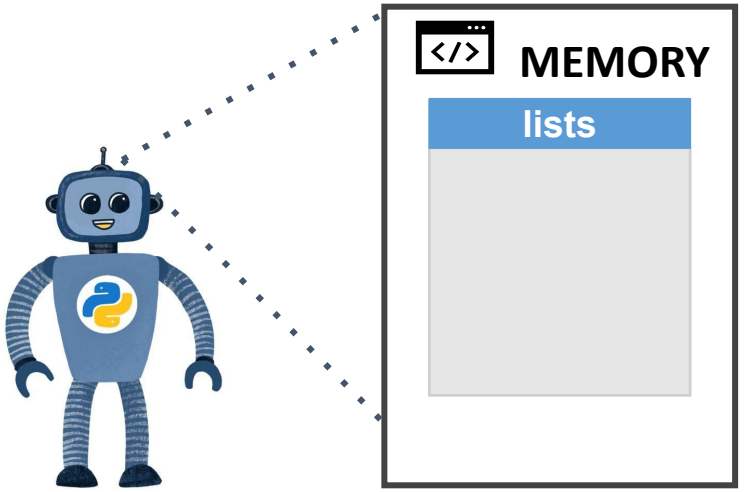
LISTS



0

pythons_list = [“Hello”, “my”, “name”, “is”, “Python”, “!”]

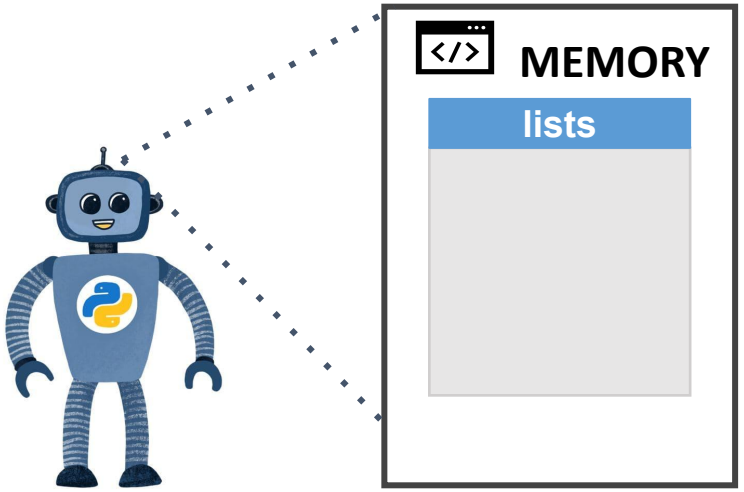
LISTS



1

pythons_list = [“Hello”, “my”, “name”, “is”, “Python”, “!”]

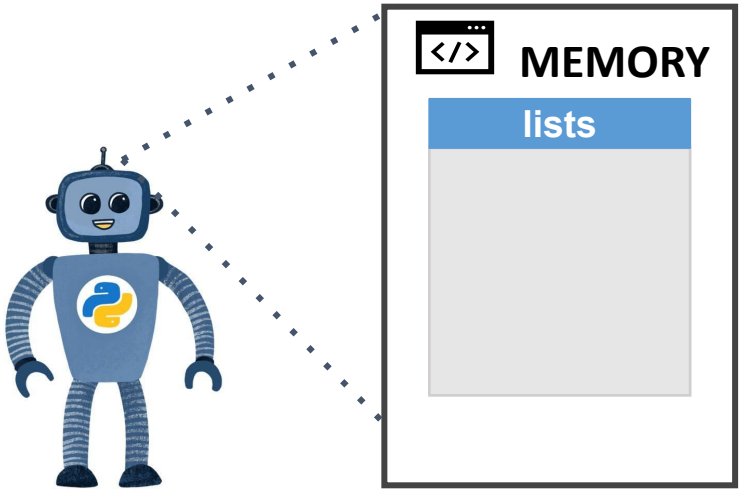
LISTS



2

pythons_list = [“Hello”, “my”, “name”, “is”, “Python”, “!”]

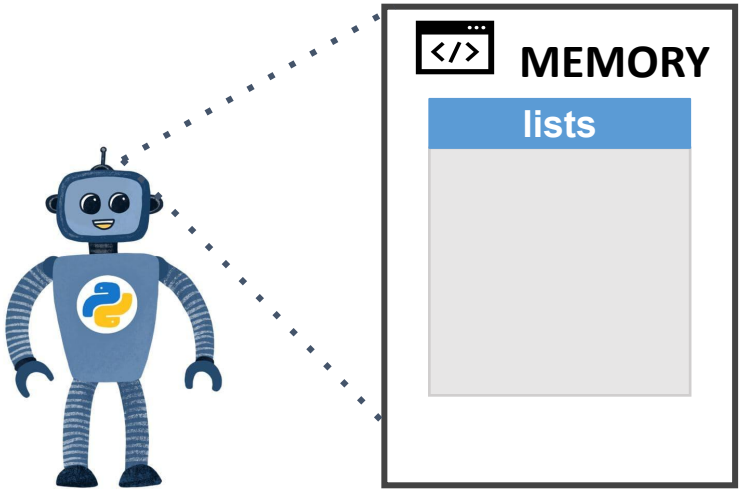
LISTS



3

pythons_list = [“Hello”, “my”, “name”, “is”, “Python”, “!”]

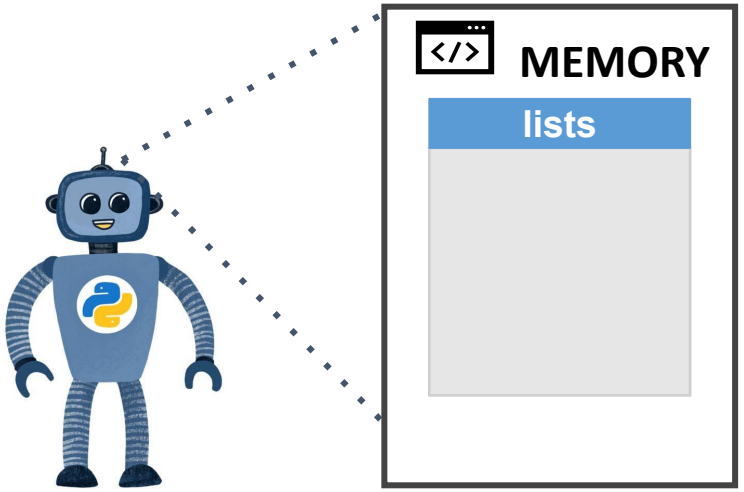
LISTS



4

pythons_list = [“Hello”, “my”, “name”, “is”, “Python”, “!”]

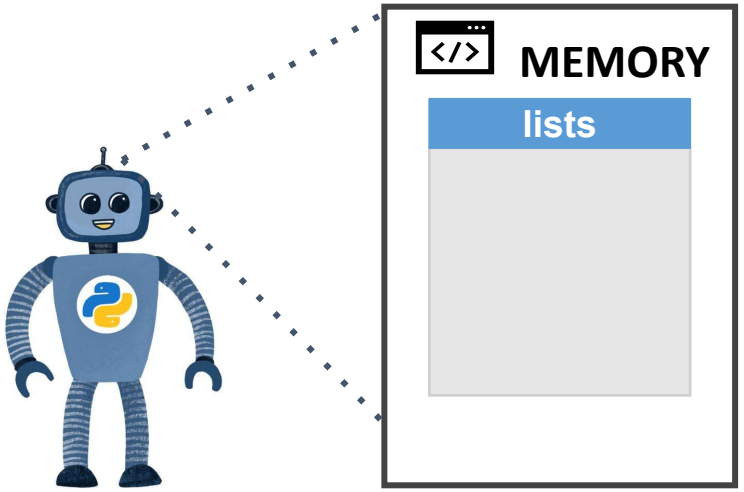
LISTS



5

pythons_list = [“Hello”, “my”, “name”, “is”, “Python”, “!”]

LISTS



Python starts counting at zero!

0	1	2	3	4	5

pythons_list = [“Hello”, “my”, “name”, “is”, “Python”, “!”]

LISTS

Python starts counting at zero!

0	1	2	3	4	5
“Hello”	“my”	“name”	“is”	“Python”	“!”

```
python_list = ["Hello", "my", "name", "is", "Python", "!"]
```

LISTS

Python starts counting at zero!

0	1	2	3	4	5

pythons_list = [“Hello”, “my”, “name”, “is”, “Python”, “!”]

pythons_list[0]
> “Hello“

LISTS

Python starts counting at zero!

0	1	2	3	4	5

pythons_list = [“Hello”, “my”, “name”, “is”, “Python”, “!”]

pythons_list[2]
> “name

LISTS

Python starts counting at zero!

0	1	2	3	4	5

pythons_list = [“Hello”, “my”, “name”, “is”, “Python”, “!”]

pythons_list[2:4]

> “name” “is” “Python”

LISTS

Python starts counting at zero!

	0	1	2	3	4	5
pythons_list =	[“Hello”,	“my”,	“name”,	“is”,	“Python”,	“!”]

pythons_list[2:4]
> “name” “is” “Python”

Lists are used to store multiple values/variables in one object

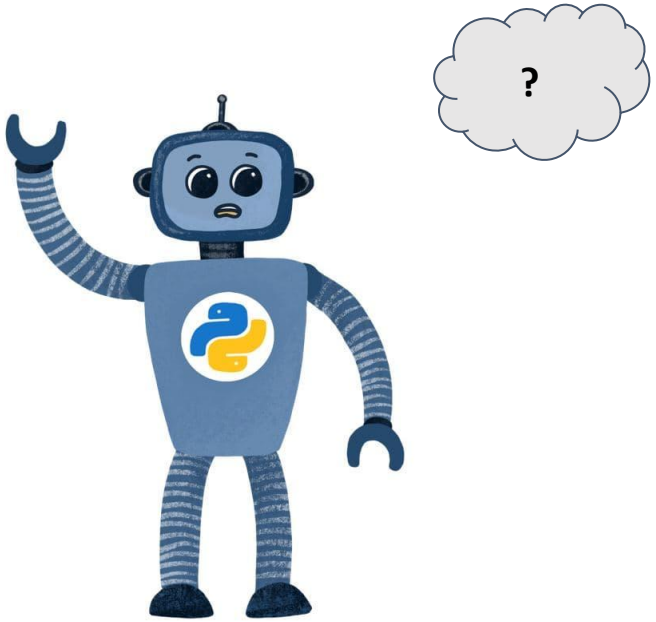
We can retrieve specific information from a list using <u>indexing</u>

COMPARISON OPERATORS

```
age_julian  = 23
age_liz = 30
```

“Python - can you answer this question for me?”

1. Are Julian and Liz the same age?



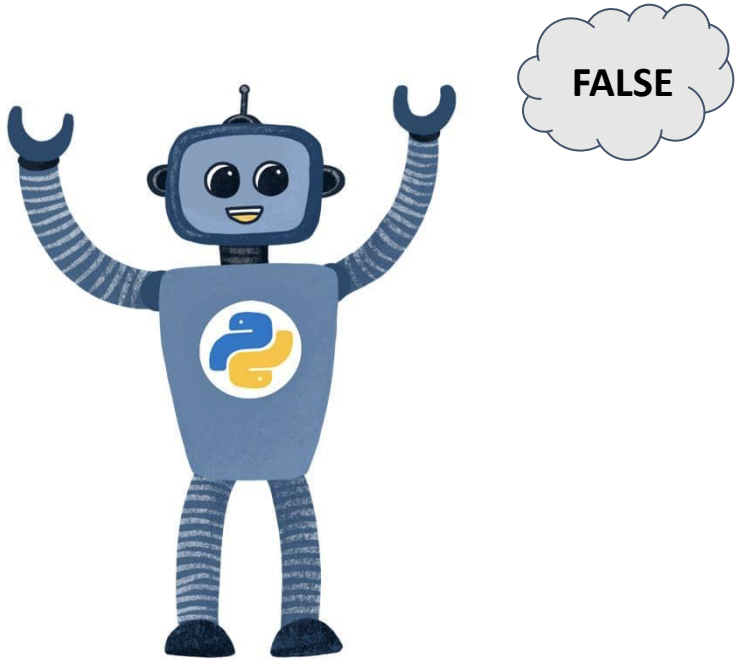
COMPARISON OPERATORS

```
age_julian  = 23
age_liz = 30
```

“Python - can you answer this question for me?”

1. Are Julian and Liz the same age?

```
age_julian == age liz
```

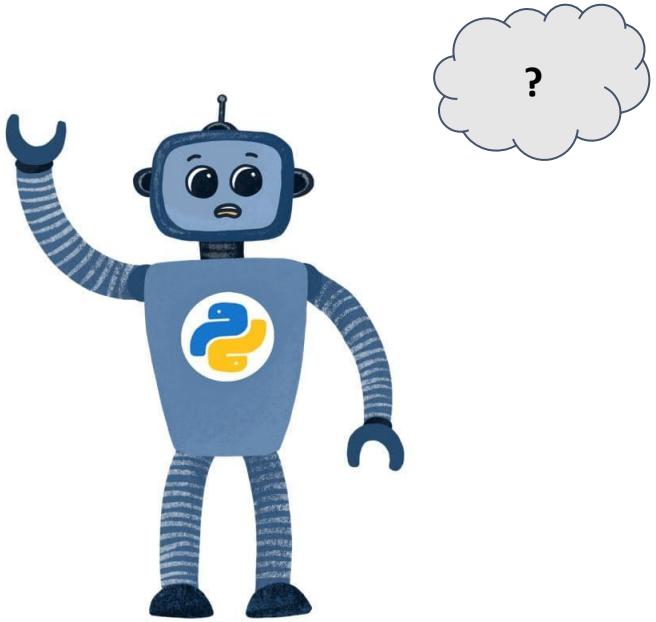


COMPARISON OPERATORS

```
age_julian  = 23
age_liz = 30
```

“Python - can you answer this question for me?”

1. Is Liz older than Julian?



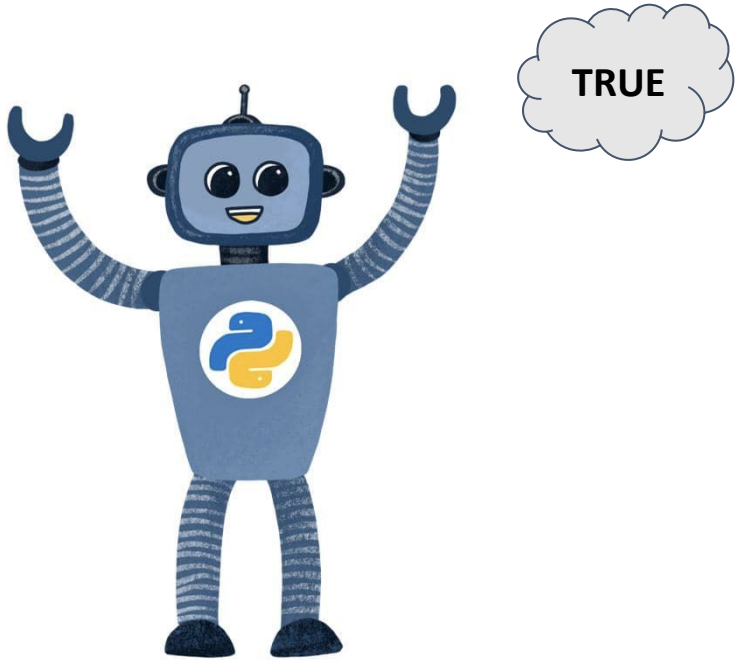
COMPARISON OPERATORS

```
age_julian  = 23
age_liz = 30
```

“Python - can you answer this question for me?”

1. Is Liz older than Julian?

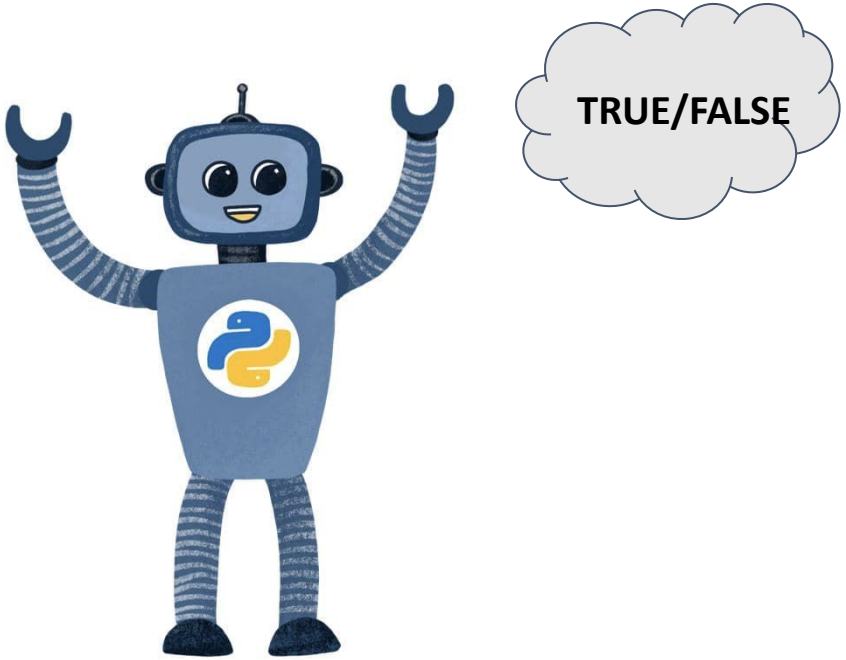
```
age_liz > age_julian
```



COMPARISON OPERATORS

“Python - can you answer this question for me?”

==	equal?
!=	not equal?
<	less?
>	greater?
<=	less or equal?
>=	greater or equal?



NOTEBOOK TIME!

PART 1

RECAP

Variable Types:

```
string = "I am a string"

integer = 42
float_  = 4.2

list_ = ["I", "am", "a", "list"]
```

Type Conversion:

```
int("42")

str(123)
```

Commenting:

This is a comment

Comparison Operators:

```
124 > 345
```

```
False
```

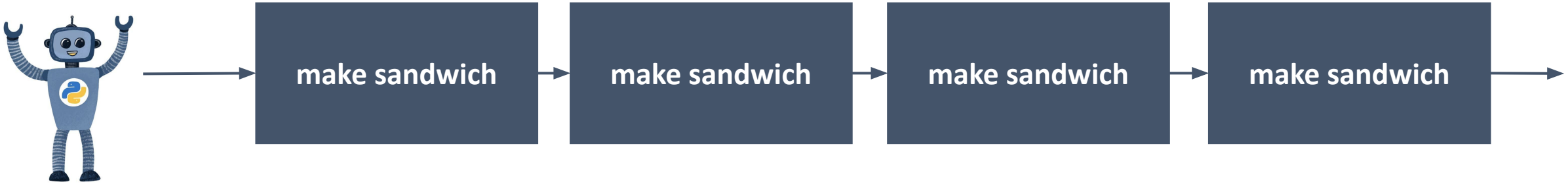
LOOPS

LOOPS

“Python - We need 4 sandwiches!”

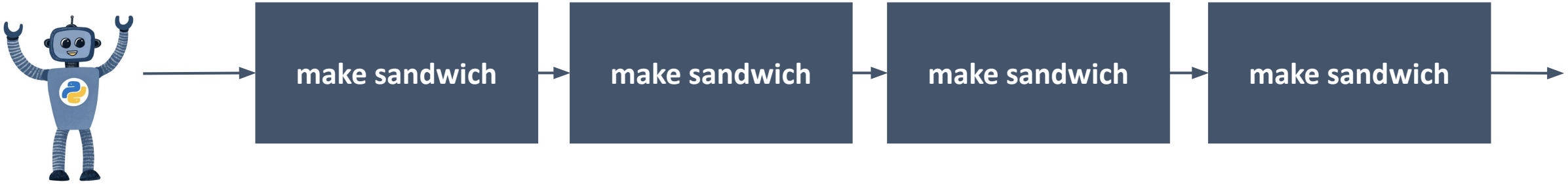
LOOPS

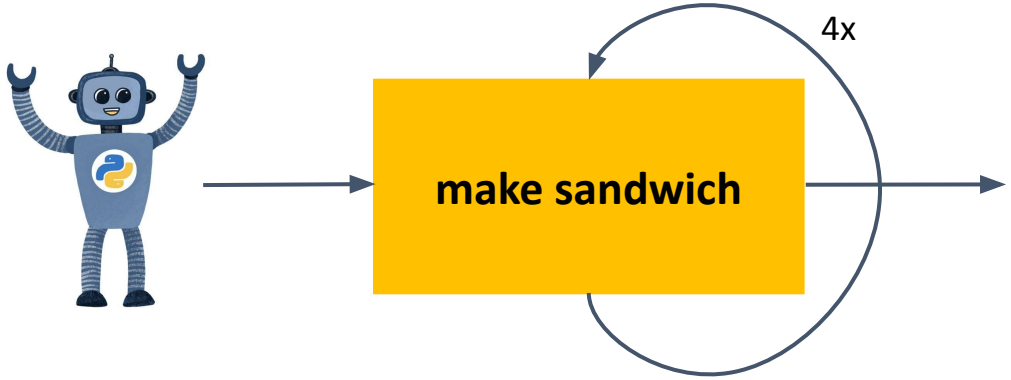
“Python - We need 4 sandwiches!”



LOOPS

“Python - We need 4 sandwiches!”

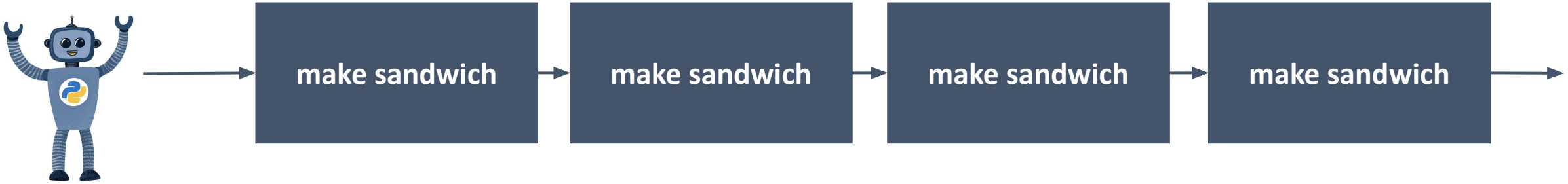


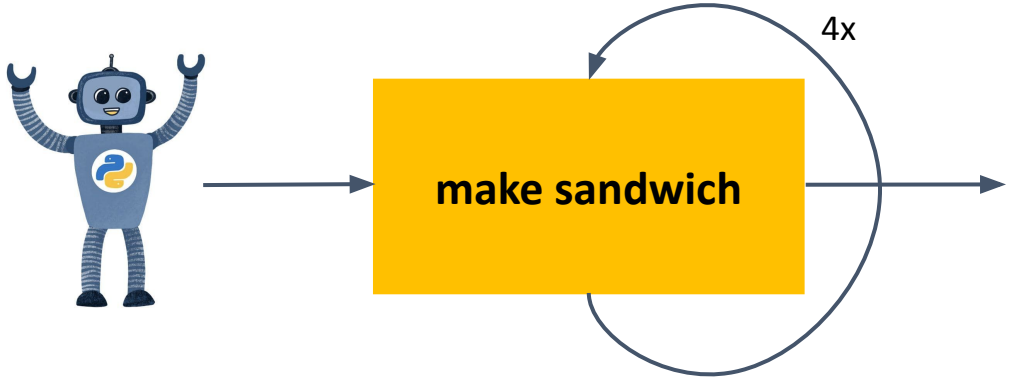


LOOPS

LOOPS ARE USED TO AVOID REPETITIONS!

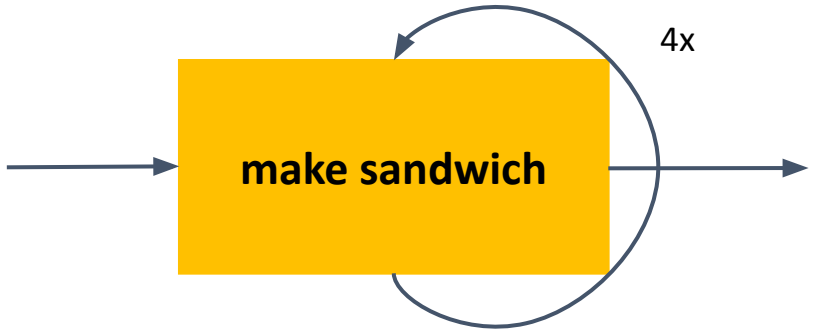
“Python - We need 4 sandwiches!”





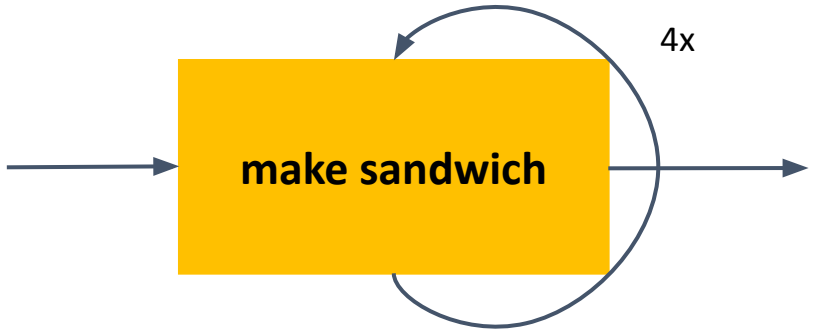
LOOPS

A: “Python - We need 4 sandwiches!”

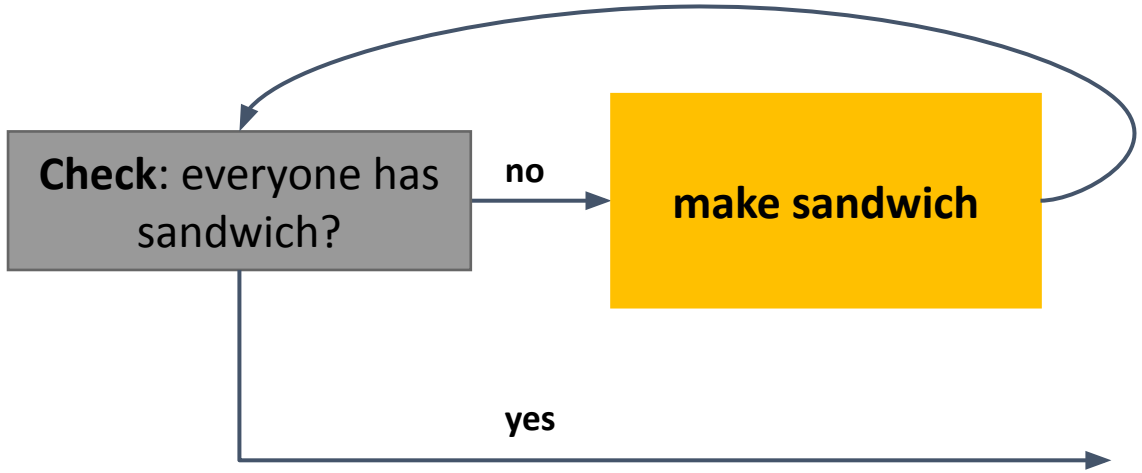


LOOPS

A: “Python - We need 4 sandwiches!”

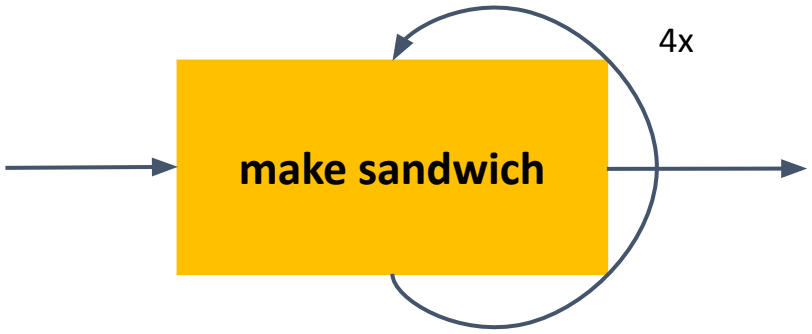


B: “We need one sandwich for every participant of this workshop!”



LOOPS - for

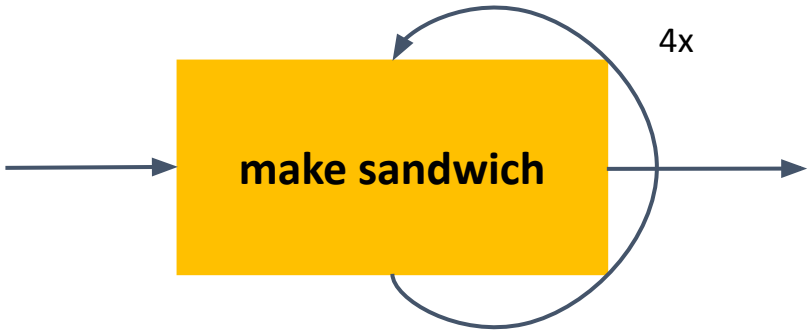
A: “Python - We need 4 sandwiches!”




```
for count in range(4):

    print(count)

    makeSandwich()


[]: 0,1,2,3
```

LOOPS - for

A: “Python - We need 4 sandwiches!”

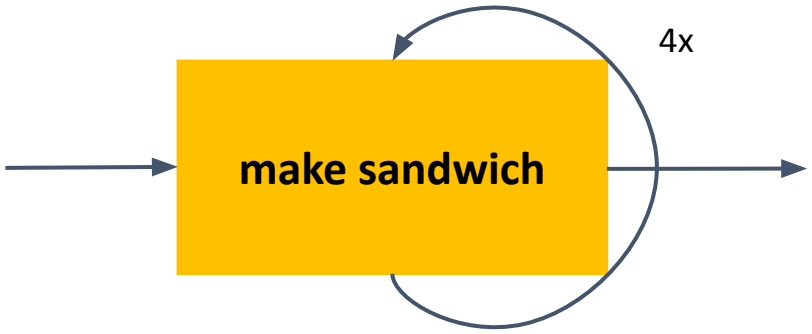


iterating variable

```
for count in range(4):

    print(count)

    makeSandwich()


[]: 0,1,2,3
```

LOOPS - for

A: “Python - We need 4 sandwiches!”



[0,1,2,3]

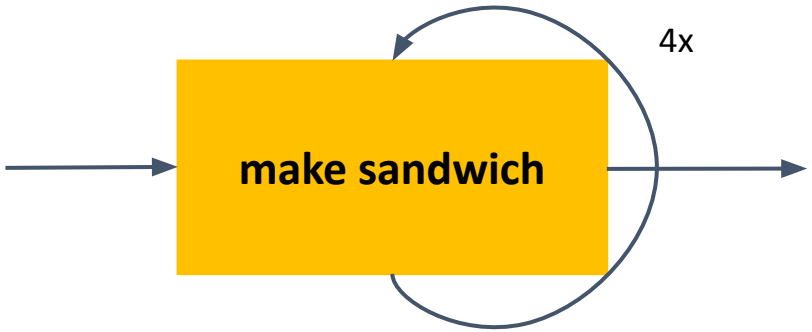
some sequence

iterating variable

```
for count in range(4):

    print(count)

    makeSandwich()


[]: 0,1,2,3
```

LOOPS - for

A: “Python - We need 4 sandwiches!”



[**0**,1,2,3]

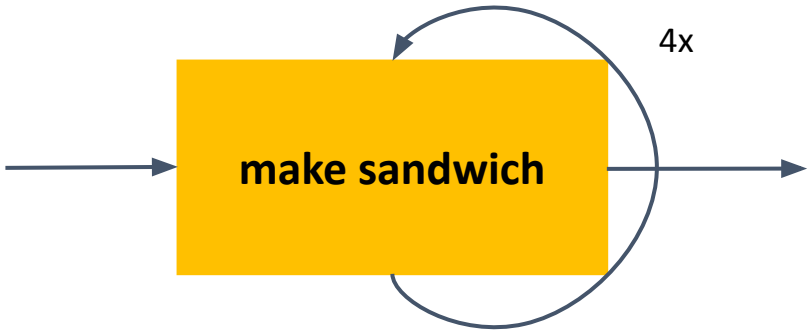
some sequence

iterating variable

```
for count in range(4):

    print(count)

    makeSandwich()


[]: 0,1,2,3
```

LOOPS - for

A: “Python - We need 4 sandwiches!”



[0,**1**,2,3]

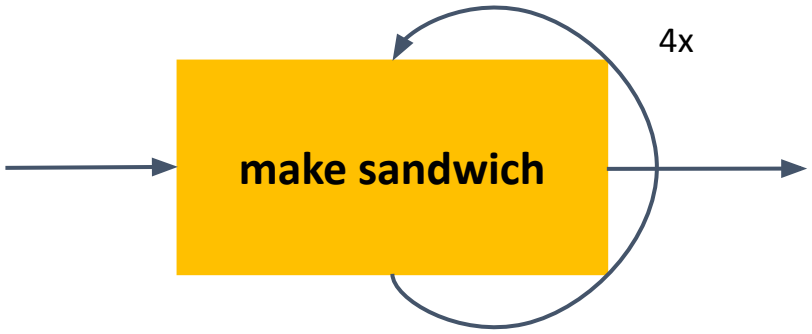
some sequence

iterating variable

```
for count in range(4):

    print(count)

    makeSandwich()


[]: 0,1,2,3
```

LOOPS - for

A: “Python - We need 4 sandwiches!”



[0,1,**2**,3]

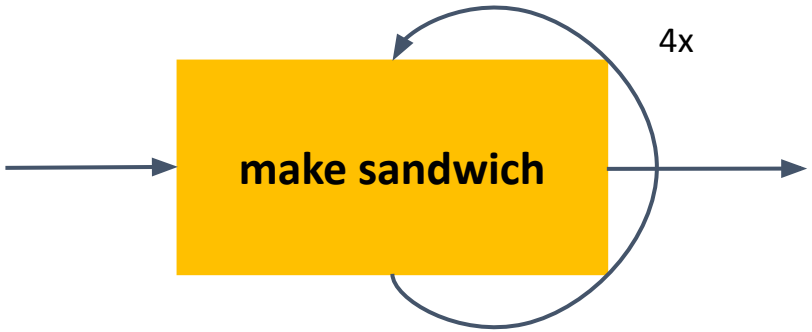
some sequence

iterating variable

```
for count in range(4):

    print(count)

    makeSandwich()


[]: 0,1,2,3
```

LOOPS - for

A: “Python - We need 4 sandwiches!”



[0,1,2,**3**]

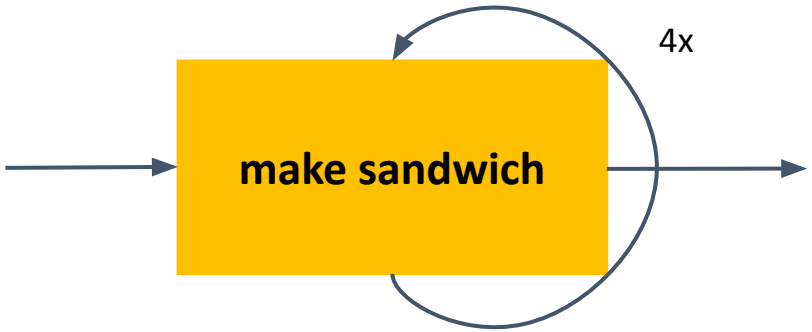
some sequence

iterating variable

```
for count in range(4):

    print(count)

    makeSandwich()


[]: 0,1,2,3
```

LOOPS - for

A: “Python - We need 4 sandwiches!”



some sequence

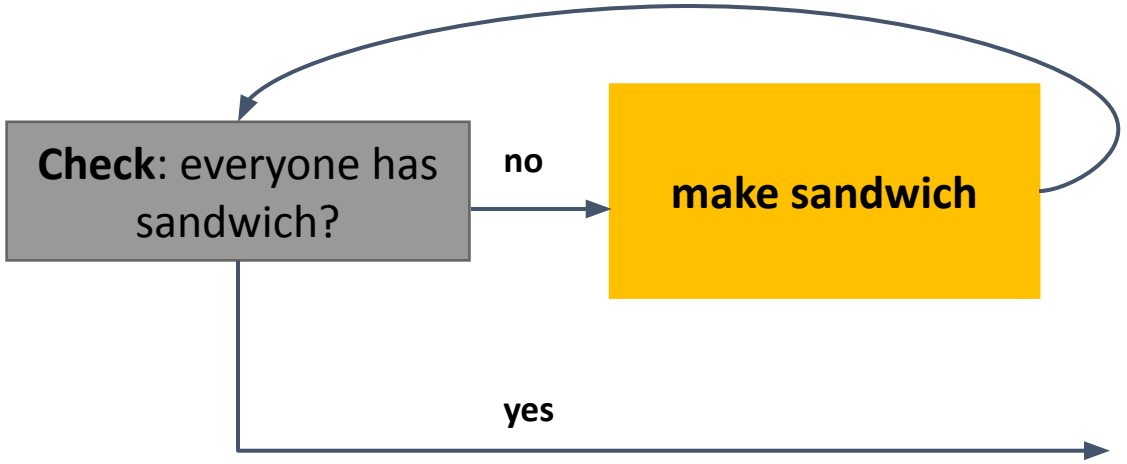
iterating variable

```
for count in range(4):

    print(count)

    makeSandwich()


[]: 0,1,2,3
```

A for loop is used for iterating over a sequence

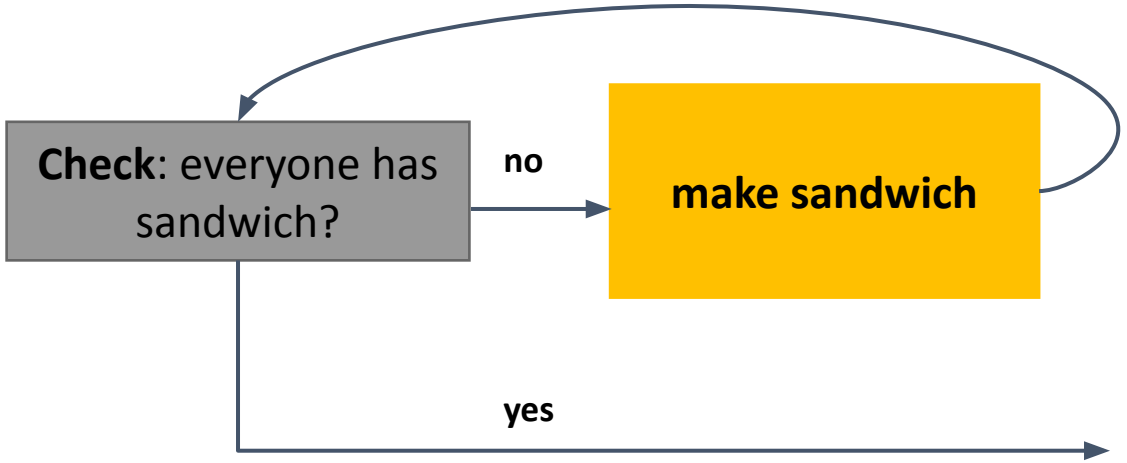
LOOPS - while

B: “We need one sandwich for every participant of this workshop!”



LOOPS - while

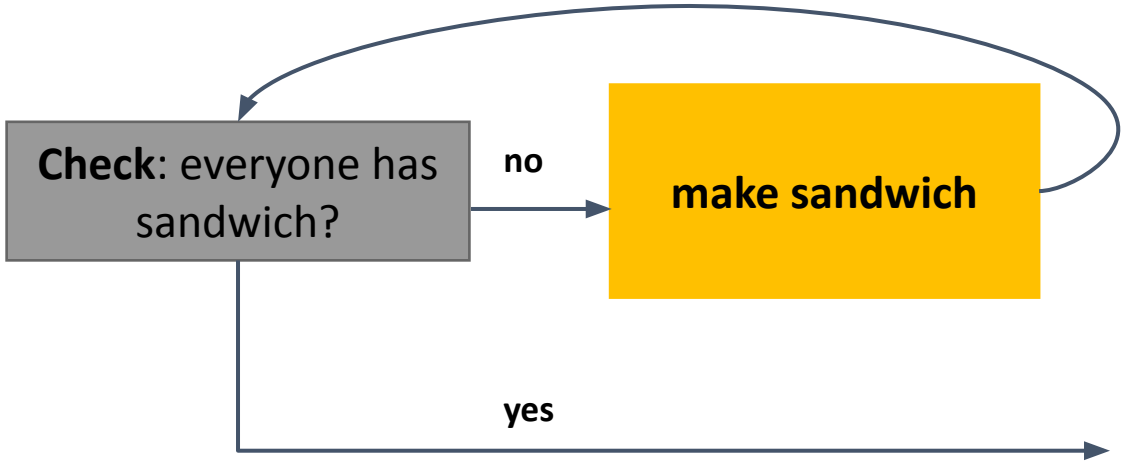
B: "We need one sandwich for every participant of this workshop!"




```python
# how many sandwiches do we need?
num_participants = 20

# how many sandwiches do we already have?
num_sandwiches = 0

# check if we need more sandwiches
while num_sandwiches < num_participants:

    makeSandwich() # make sandwich

    # count new sandwich
    num_sandwichs = num_sandwichs + 1
```

LOOPS - while

B: "We need one sandwich for every participant of this workshop!"




```
# how many sandwiches do we need?
num_participants = 20

# how many sandwiches do we already have?
num_sandwiches = 0

# check if we need more sandwiches
while num_sandwiches < num_participants:

    makeSandwich() # make sandwich

    # count new sandwich
    num_sandwichs = num_sandwichs + 1
```

NOTEBOOK TIME!

PART 2

RECAP

for loops:

```
for counter in range(3):
```

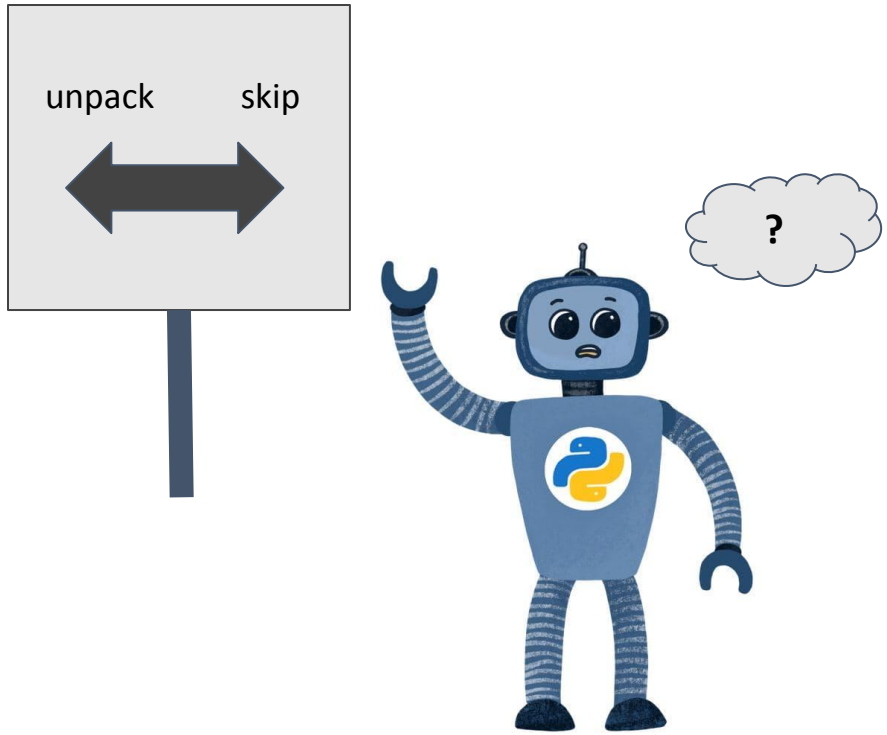
while loops:

```
while count > 5:
```

FLOW CONTROL

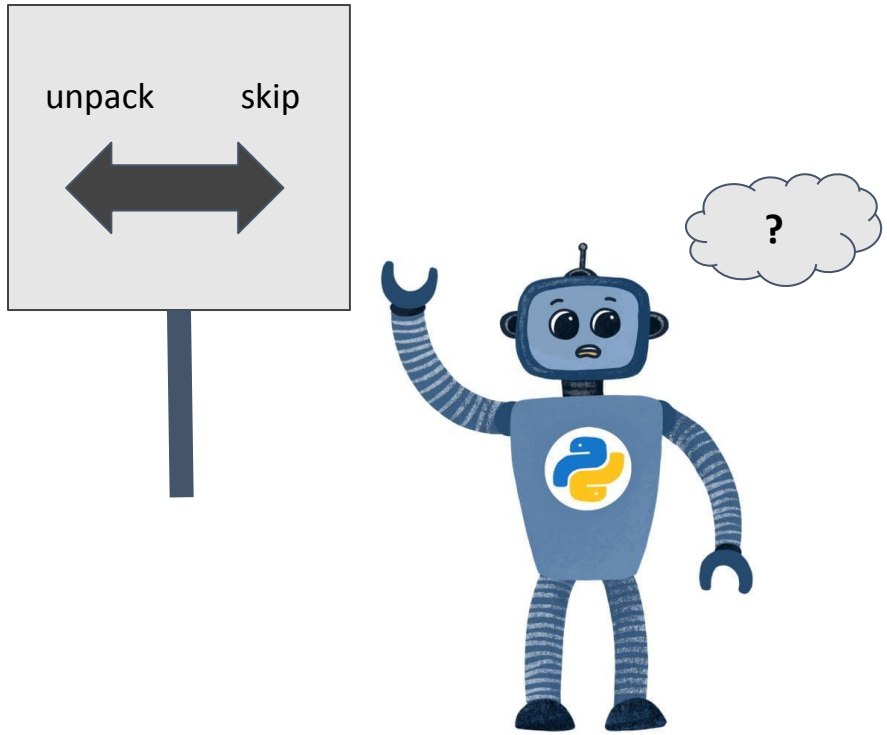
FLOW CONTROL

“Python - unpack the bread… but <u>only if</u> it’s still in its packaging!”



FLOW CONTROL

“Python - unpack the bread… but <u>only if</u> it’s still in its packaging!”

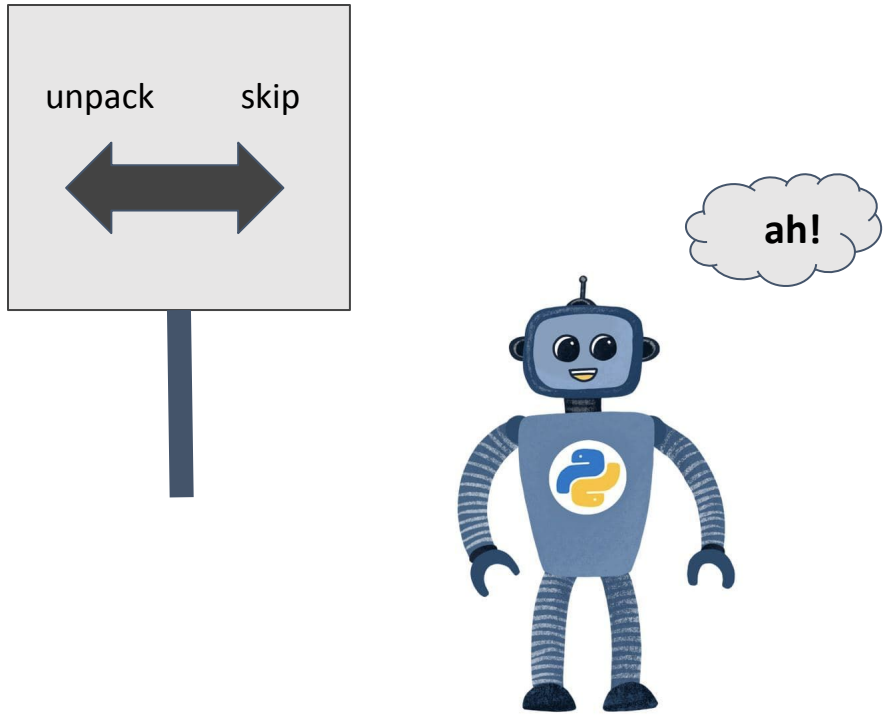


A

B

FLOW CONTROL

“Python - unpack the bread… but <u>only if</u> it’s still in its packaging!”

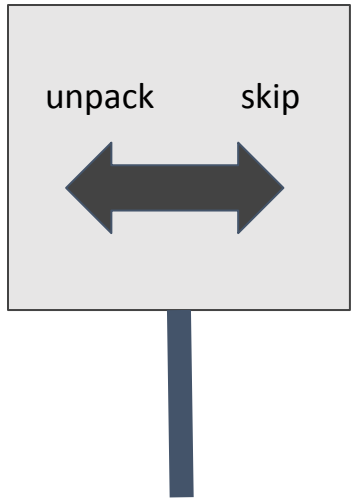


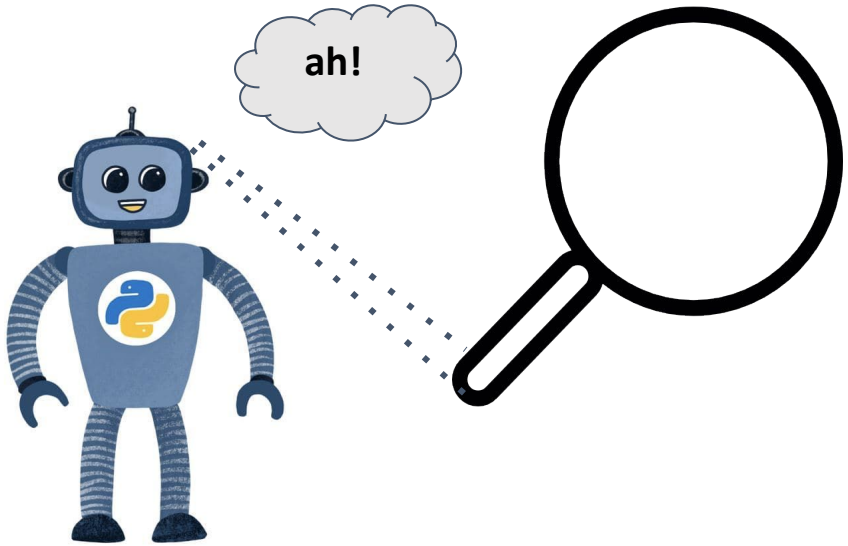
A **B**

➢ **Check for packaging:**

FLOW CONTROL

“Python - unpack the bread… but <u>only if</u> it’s still in its packaging!”



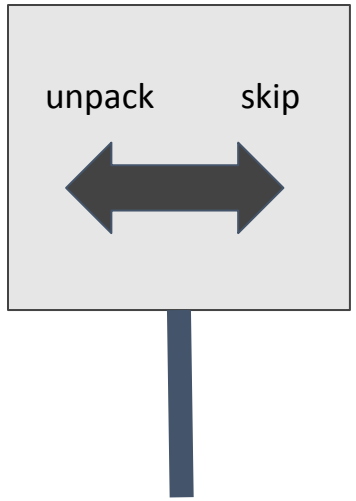


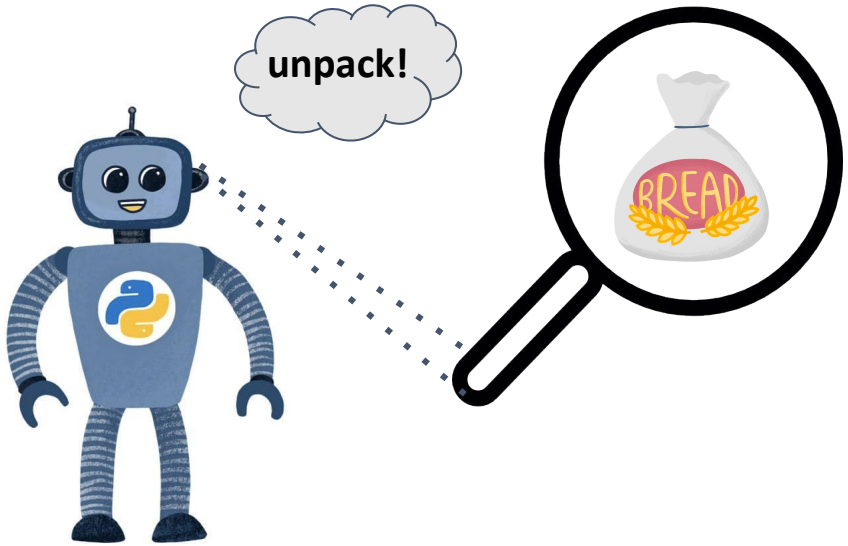
A **B**

➢ **Check for packaging:**

FLOW CONTROL

“Python - unpack the bread… but <u>only if</u> it’s still in its packaging!”



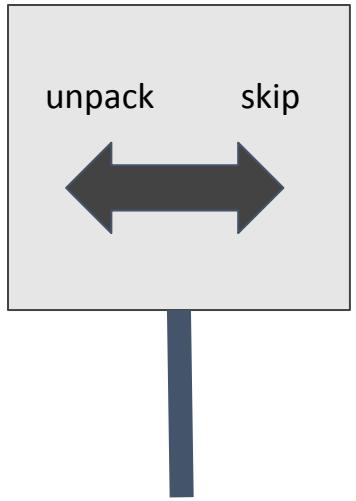


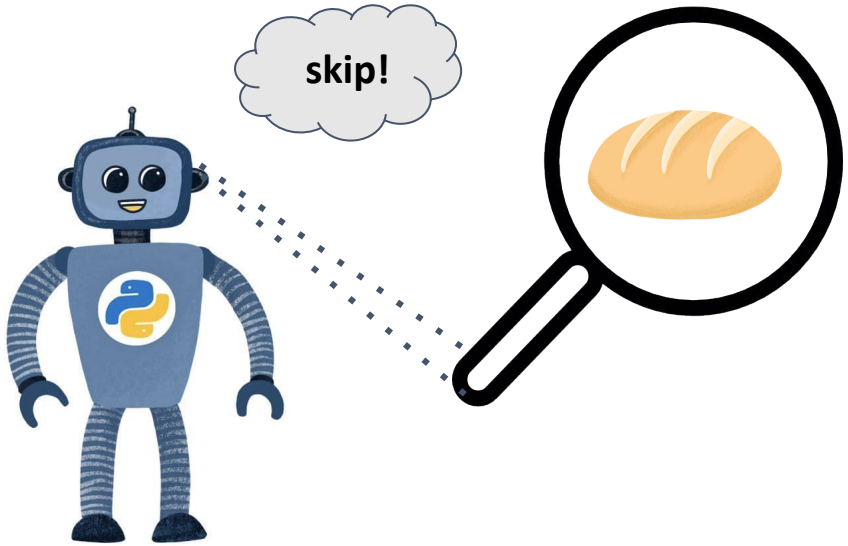
A **B**

➢ **Check for packaging:**

FLOW CONTROL

“Python - unpack the bread… but <u>only if</u> it’s still in its packaging!”

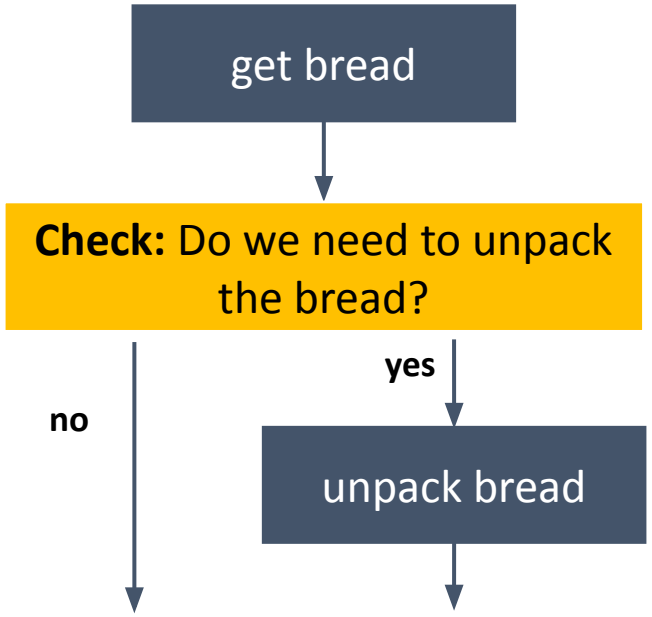




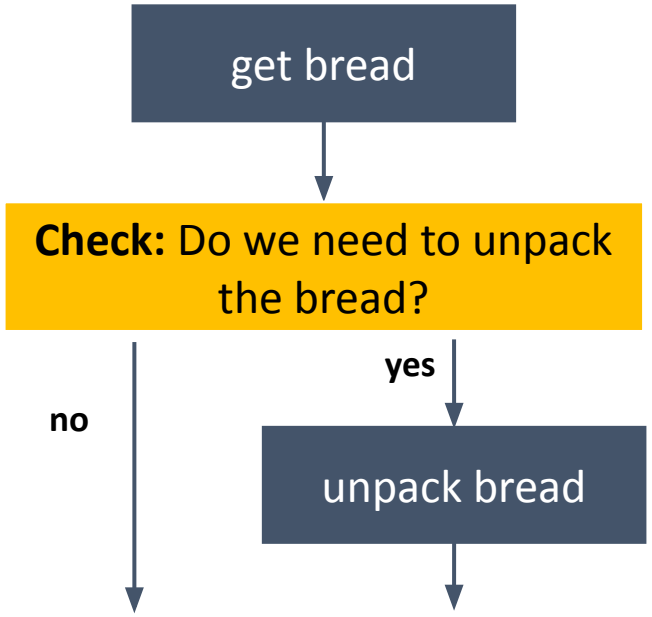
A **B**

➢ **Check for packaging:**

FLOW CONTROL - if statements

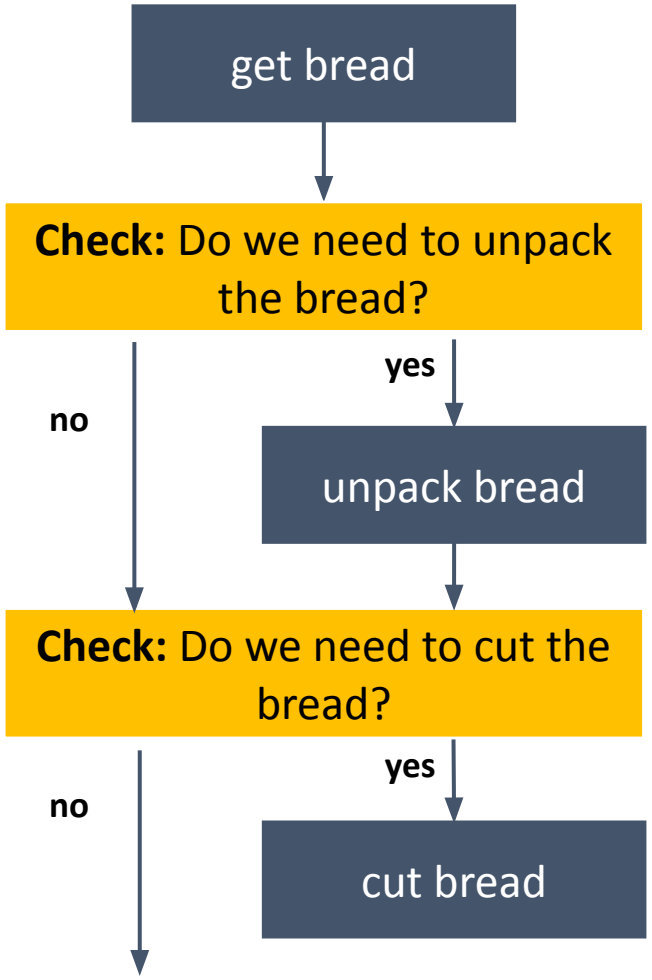


FLOW CONTROL - if statements




```
# get bread for sandwich
getBread()

# in case the bread is wrapped:
if bread == wrapped:

    unpackBread() # unpack
```

FLOW CONTROL - if statements




```
# get bread for sandwich
getBread()

# in case the bread is wrapped:
if bread == wrapped:

    unpackBread() # unpack

# in case bread is not cut yet:
if bread != cut:

    cutBread() # cut
```

NOTEBOOK TIME!

PART 3:

RECAP

General Structure of if-elif-else:

```
if this == true:
    doThis()

elif that == true:
    doThat()

else:
    doSomethingElse()
```

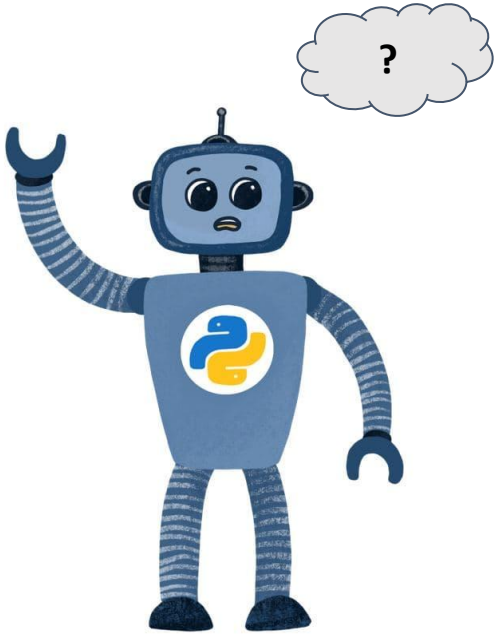
Combining Loops and if-statements:

```
for element in listOfThings:

    if element == true:
        doSomething()
```

FUNCTIONS

FUNCTIONS

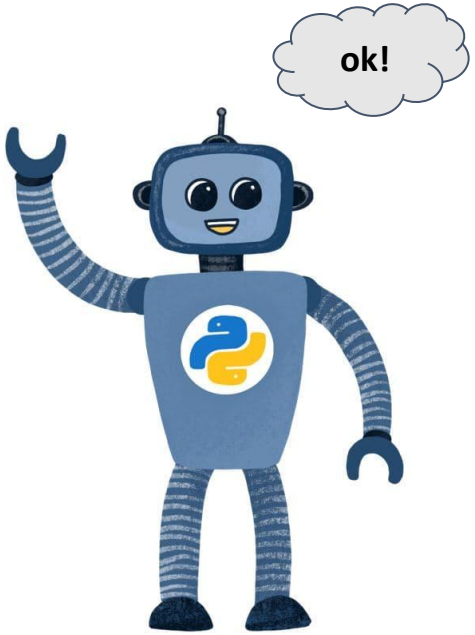
“Python - Put butter on bread!”



FUNCTIONS

“Python - Put butter on bread!”

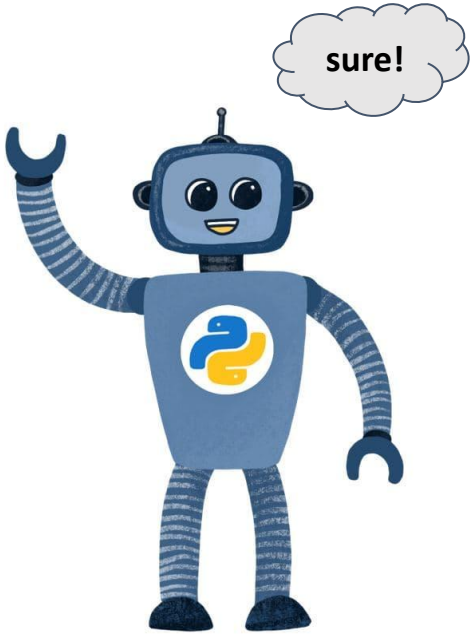
- get ingredients and tools:
  - butter
  - bread
  - knife
- check: Is bread cut? If not, cut bread!
- put slice of bread on plate
- grab knife
- use knife to get 5g butter
- spread butter on slice of bread evenly
- put down knife



FUNCTIONS

“Python - Put butter on bread!”

- get ingredients and tools:
  - butter
  - bread
  - knife
- check: Is bread cut? If not, cut bread!
- put slice of bread on plate
- grab knife
- use knife to get 5g butter
- spread butter on slice of bread evenly
- put down knife



“Python - Put butter on the next bread!”

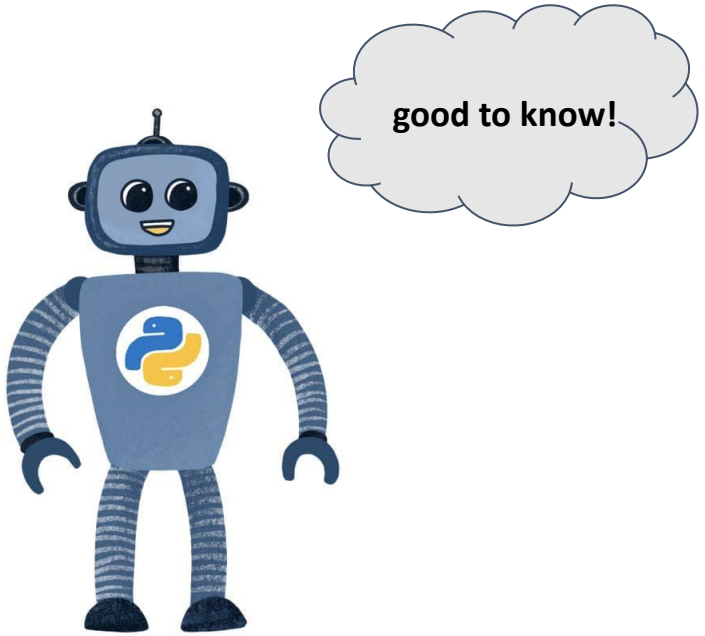
- get ingredients and tools:
  - butter
  - bread
  - knife
- check: Is bread cut? If not, cut bread!
- put slice of bread on plate
- grab knife
- use knife to get 5g butter
- spread butter on slice of bread evenly
- put down knife

FUNCTIONS

“Python - This is how you put butter on a bread!”

spreadButter()

- get ingredients and tools:
  - butter
  - bread
  - knife
- check: Is bread cut? If not, cut bread!
- put slice of bread on plate
- grab knife
- use knife to get 5g butter
- spread butter on slice of bread evenly
- put down knife



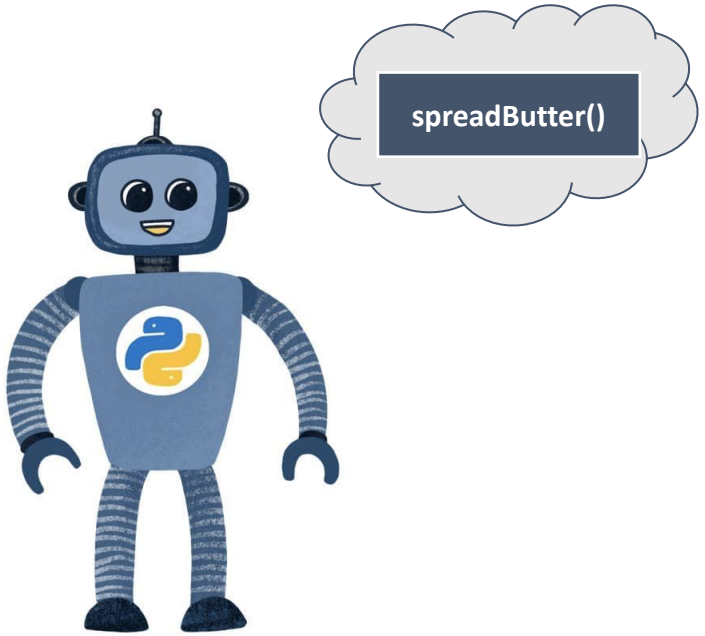
FUNCTIONS

“Python - This is how you put butter on a bread!”

spreadButter()

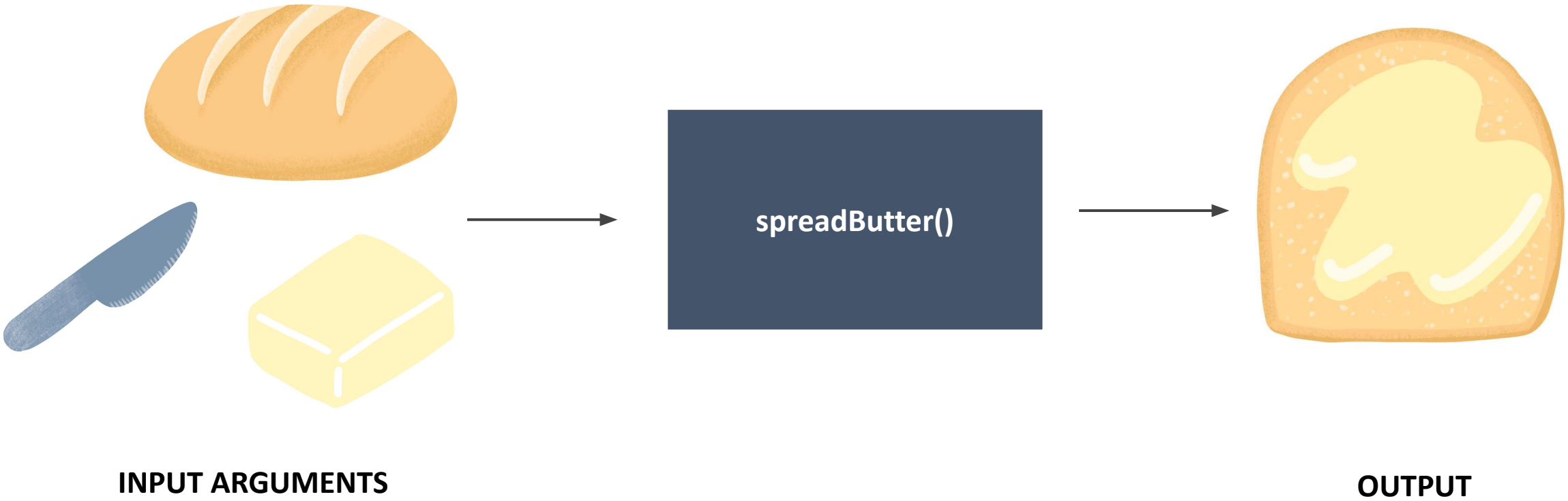
- get ingredients and tools:
  - butter
  - bread
  - knife
- check: Is bread cut? If not, cut bread!
- put slice of bread on plate
- grab knife
- use knife to get 5g butter
- spread butter on slice of bread evenly
- put down knife

“Python - Put butter on bread!”

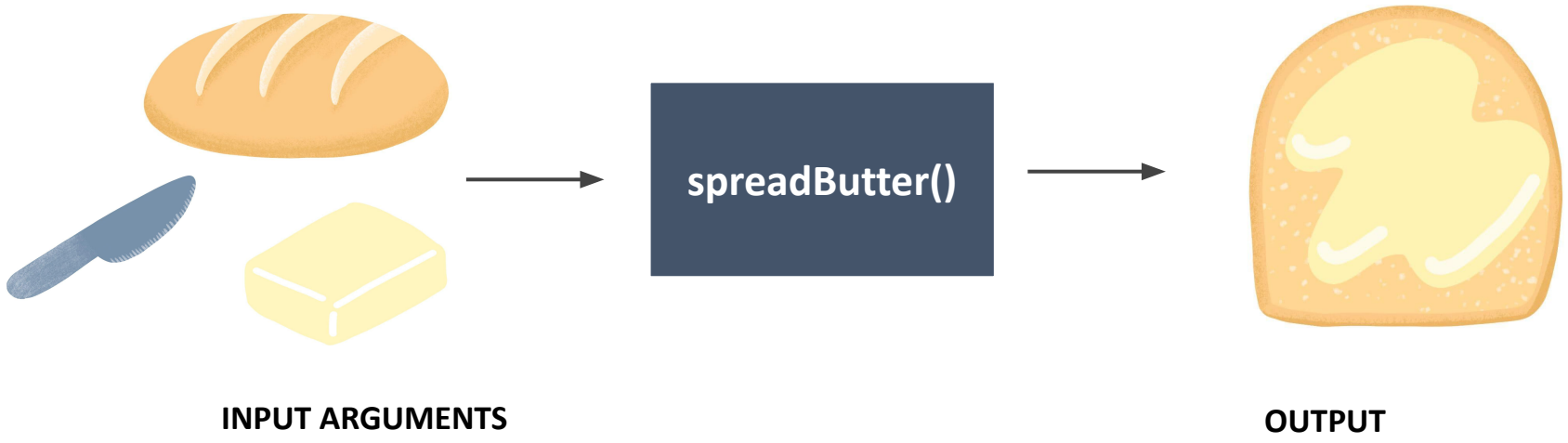


“Python - Put butter on the next bread!”

FUNCTIONS

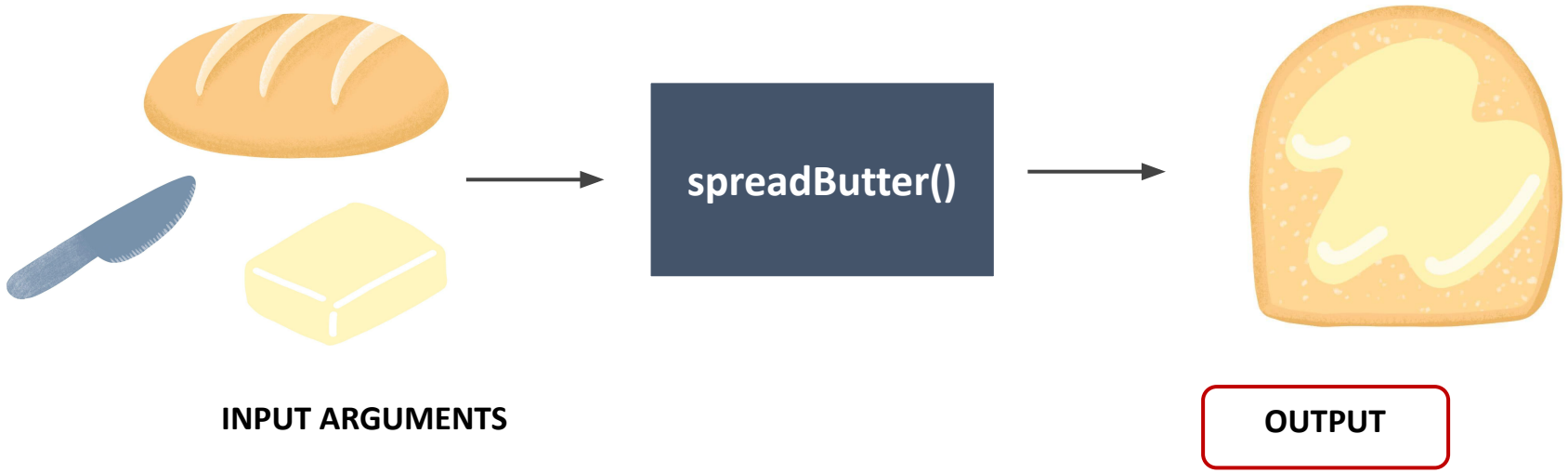


FUNCTIONS



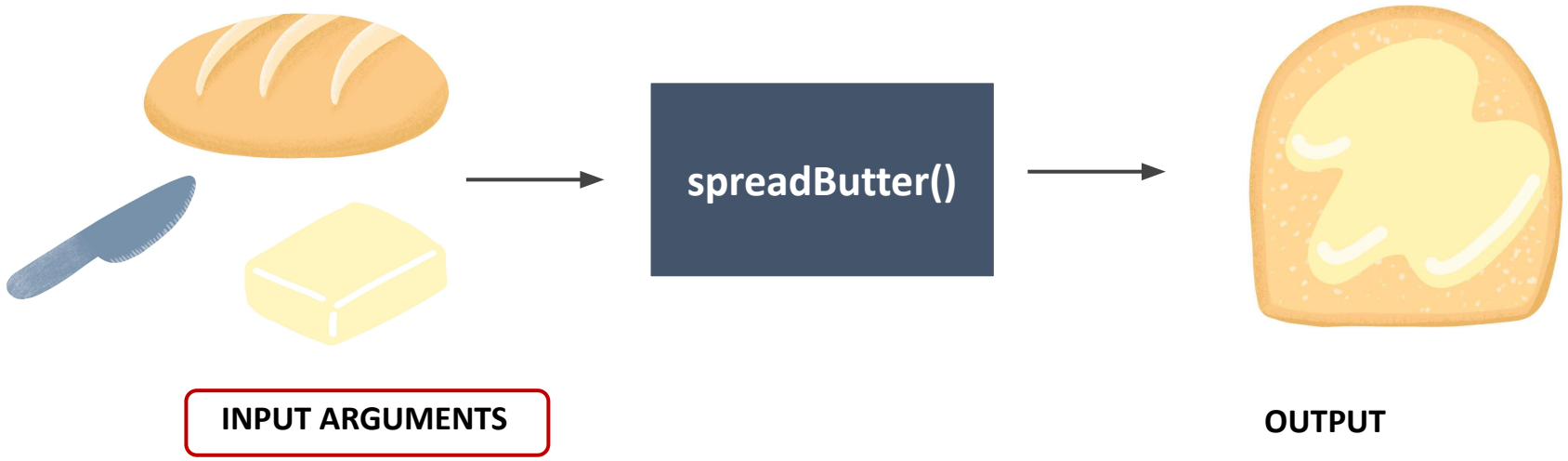

```
bread_with_butter = spreadButter(knife = myKnife, butter = REWE_butter, bread = toast)
```

FUNCTIONS




```
bread_with_butter = spreadButter(knife = myKnife, butter = REWE_butter, bread = toast)
```

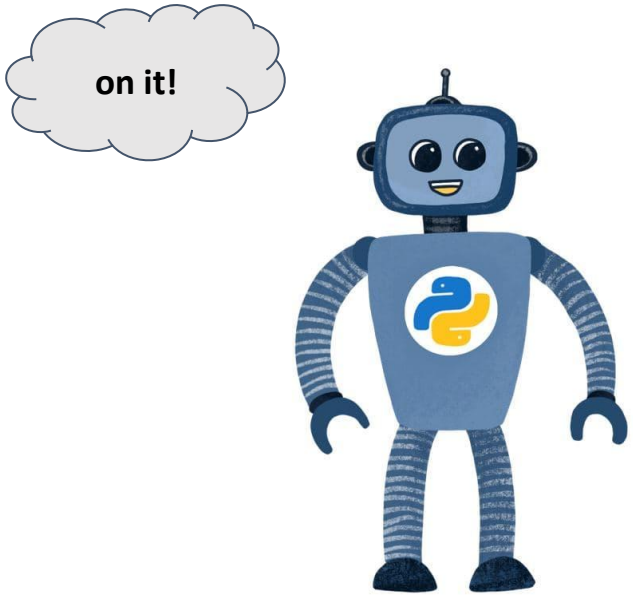
FUNCTIONS




```
bread_with_butter = spreadButter(knife = myKnife, butter = REWE_butter, bread = toast)
```

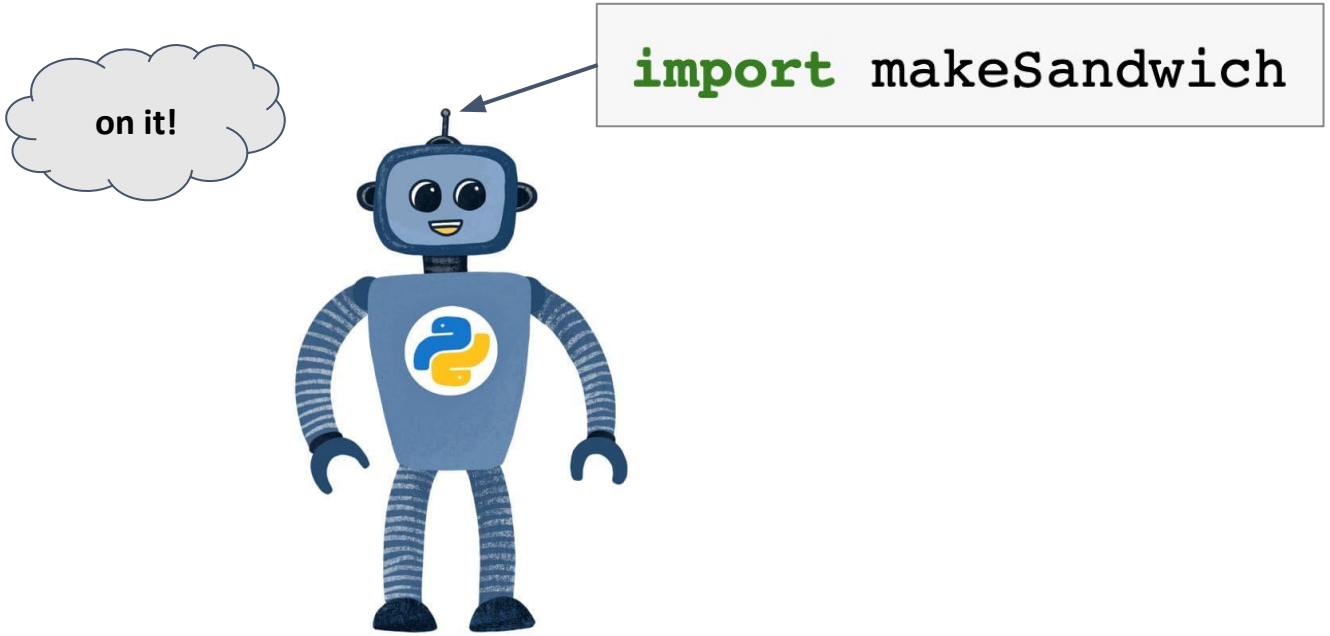
LIBARIES/MODULES

“Python - prepare to make some sandwiches!”



LIBARIES/MODULES

“Python - prepare to make some sandwiches!”



LIBARIES/MODULES

“Python - prepare to make some sandwiches!”


```
import makeSandwich
```

makeSandwich

spreadButter()

getPlate()

OpenCheese()

NOTEBOOK TIME!

PART 4

RECAP

defining functions:

```
def functionName(inputArgument):

    outputArgument = doSomethingWith(inputArgument)

    return outputArgument
```

importing libraries:

```
import random
```