CITY UNIVERSITY
LONDON

Introduction to programming using Python

Session 3

Matthieu Choplin

matthieu.choplin1@city.ac.uk

http://moodle.city.ac.uk/

CITY UNIVERSITY LONDON

Objectives

- Looping with while
- Looping with for

Motivation

On one of our previous program, we asked the user to enter a password.

- If the password was correct, we printed "Access Granted"
- Else, we printed, "Forbidden"

```
PASSWORD = 'super_password123'
password_entered = input("Enter the password: ")
if password_entered == PASSWORD:
    print("Access Granted")
else:
    print("Forbidden")
```

Motivation

However, the user only had *one chance* to enter a correct password. If the password was incorrect or correct, the program would stopped.

What if we want to make the user able to try more than once to enter a correct password?

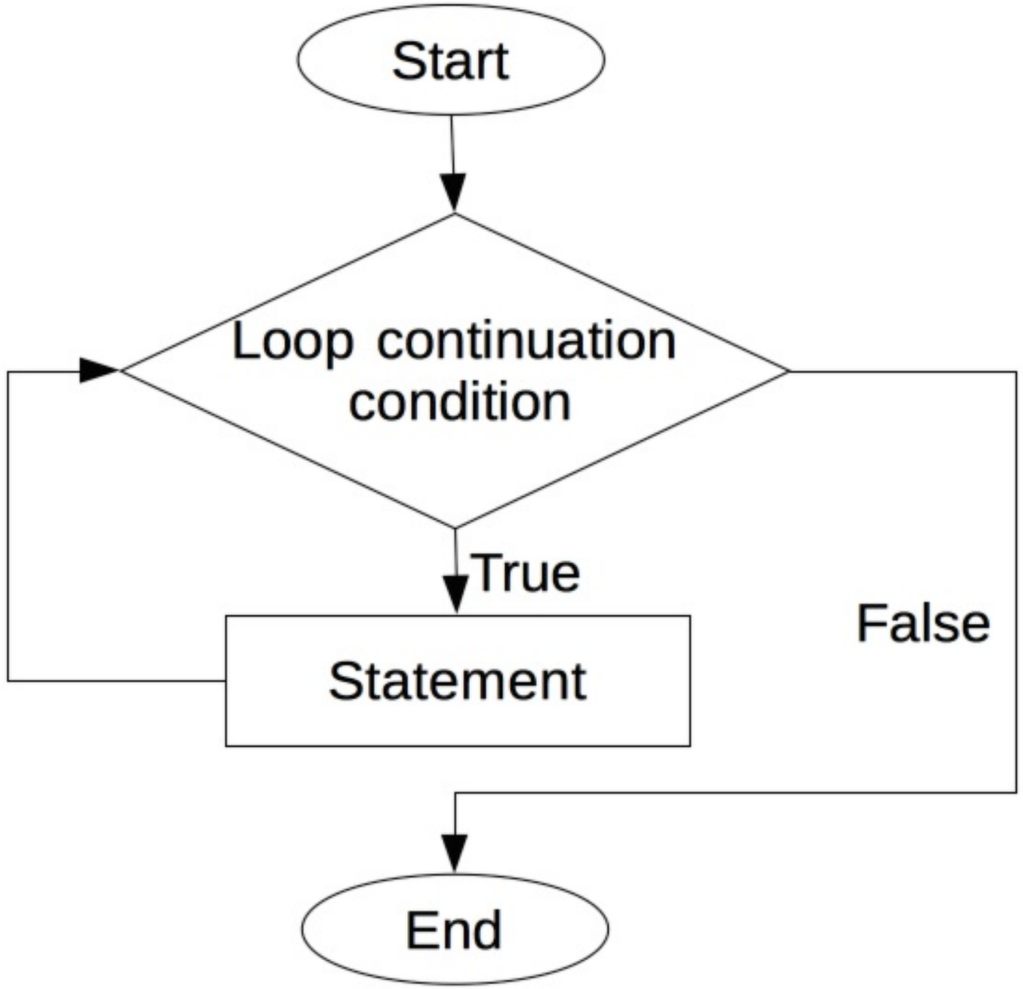
Structure of the while loop

```
while condition:
    # statement
```

Where:

- The **condition** is an expression that take the value True or False (boolean)
- The **statement** does something, mind the **indentation**
- **While** the condition is True, the **statement** or **body** of the **loop** is executed
- Each time that the body of the loop is executed is an **iteration**

Structure of the while loop, flow chart



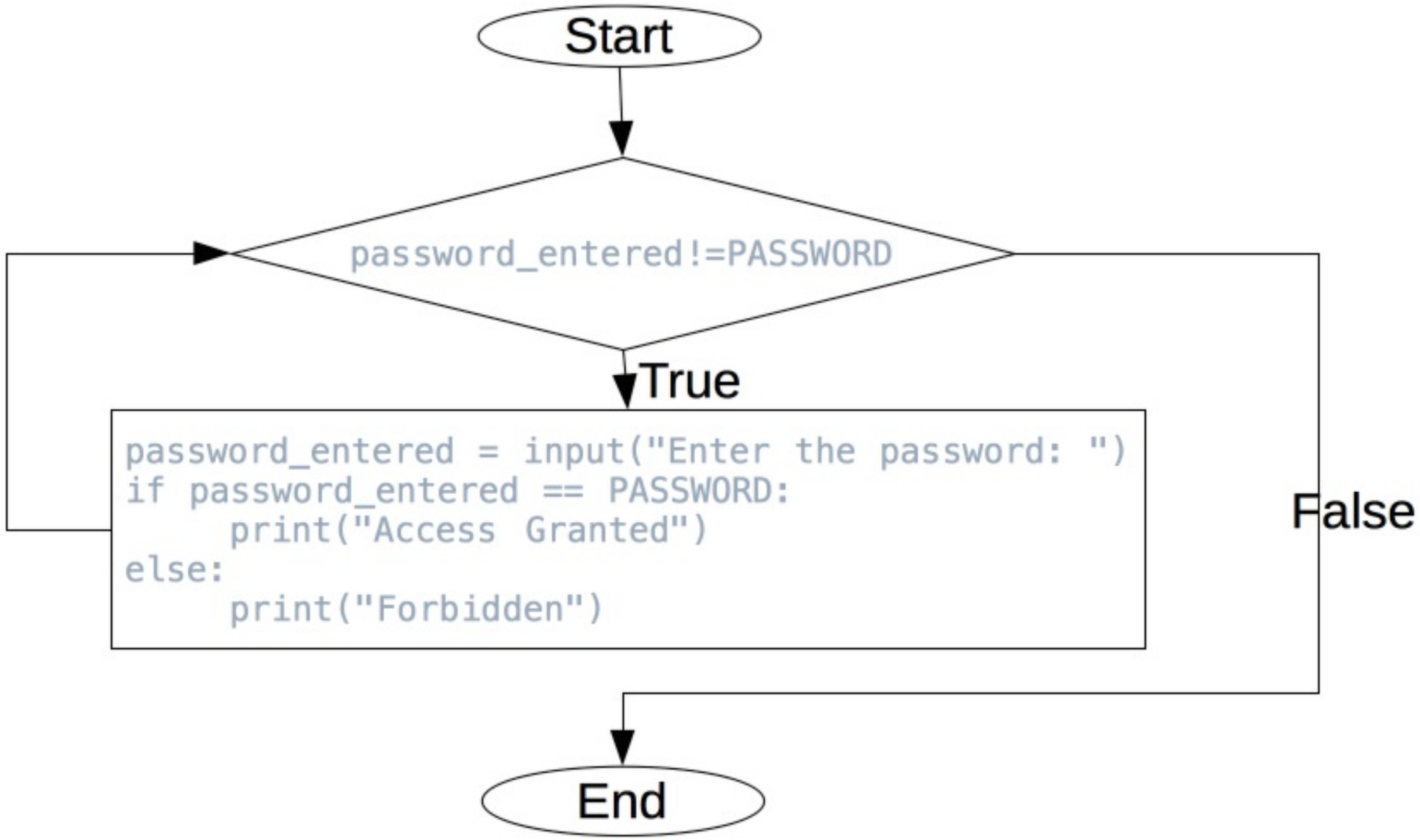
The while loop applied to our problem

```
PASSWORD = 'super_password123'
password_entered = ''
while password_entered!=PASSWORD:
    password_entered = input("Enter the password: ")
    if password_entered == PASSWORD:
        print("Access Granted")
    else:
        print("Forbidden")
```

Where:

- The **condition** is the boolean value given by the comparison of the password_entered compared to PASSWORD

Structure of the while loop, flow chart



Reminder: using variable

You cannot used a variable that has not been declared

```
PASSWORD = 'super_password123'
while password_entered!=PASSWORD:
    password_entered = input("Enter the password: ")
    if password_entered == PASSWORD:
        print("Access Granted")
    else:
        print("Forbidden")
```

Can you see why this is wrong? Try to run this program. See the error and explain what you need to correct.

Reminder: using variable

You need to declare the variable *password_entered* before using it, else, you get:

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

```
PASSWORD = 'super_password123'
password_entered = ''
while password_entered!=PASSWORD:
    password_entered = input("Enter the password: ")
    if password_entered == PASSWORD:
        print("Access Granted")
    else:
        print("Forbidden")
```

CITY UNIVERSITY LONDON

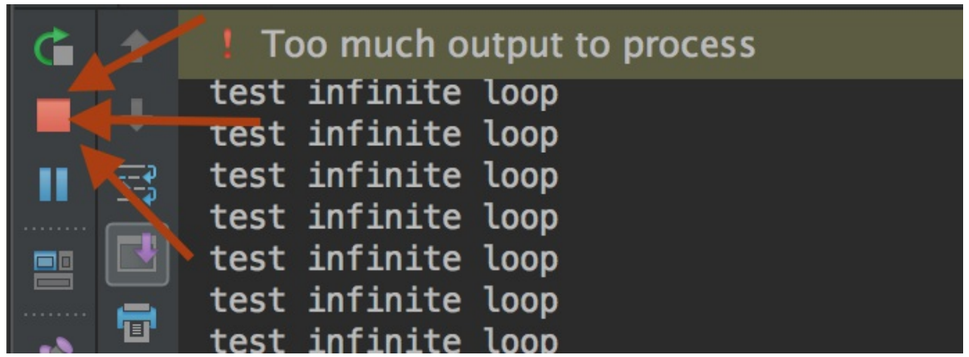
Reminder: conditions

Conditions are expressions, they return a value that can only be True or False.

A condition that is always True, if used in a while loop, produces an **infinite loop.**

```
while True:
    print('test infinite loop')
```

NB: to stop the infinite loop in Pycharm, click on the little red square:



How to avoid infinite loop

Make sure that the condition gets False at some point during the execution of the program

You can implement a counter, to limit the number of **iterations**:

```
counter=0
while counter < 5:
    counter = counter + 1 # that you can also write counter+=1
    print('test infinite loop')
```

NB: `counter = counter + 1` is equivalent to `counter += 1`

We say that we **increment** the counter at each **iteration**

Exercise: Guess Number

Make a program to ask the user to guess a the number that has been randomly generated.

Start from this file: GuessNumber.py (right click and save as)

- The user will be able to try continuously until he finds the correct number.
- The program will stop as soon as the number is found, i.e. as soon as the random number matches the entered number
- At each iteration, i.e. each time the user try a number and press enter, the program will say if the number is too high, too low or correct

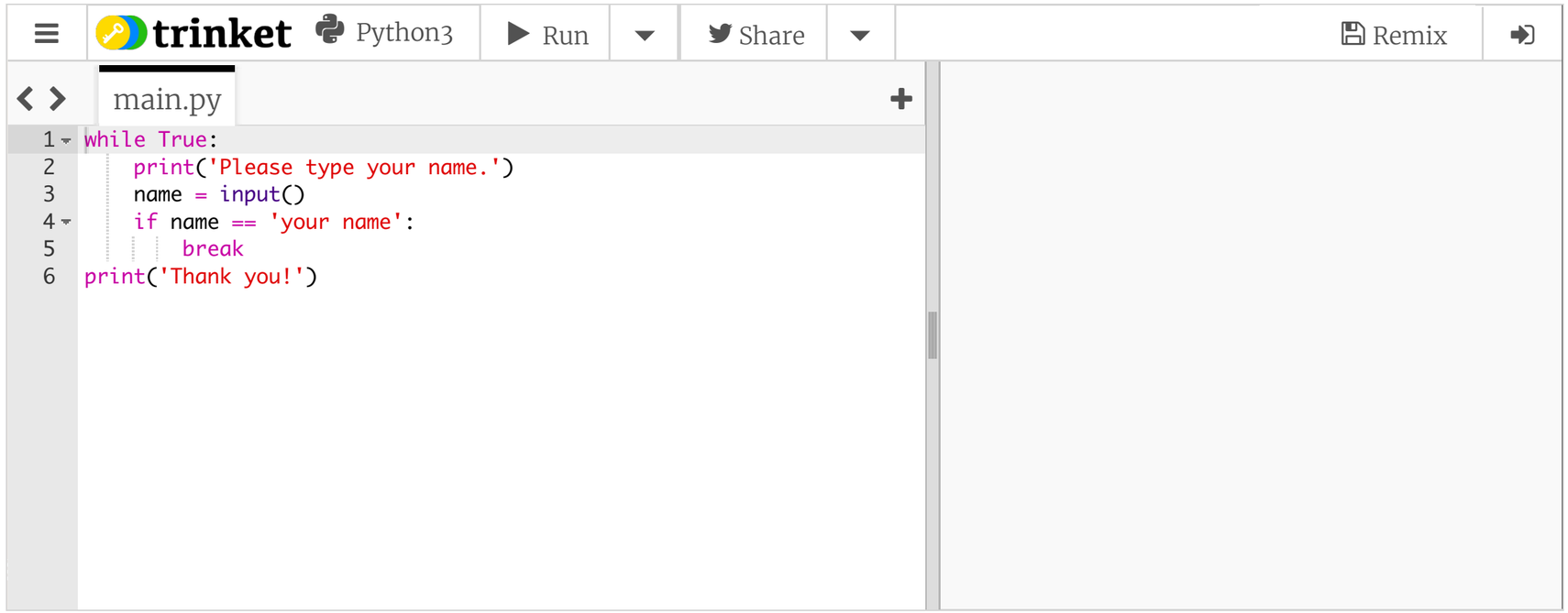
Solution: Guess Number

```
import random

# Generate a random number to be guessed
number = random.randint(1, 100)
print("Guess a magic number between 0 and 100")
guess = -1
while guess != number:
    guess = int(input("Enter your guess: "))
    if guess == number:
        print("Yes, the number is", number)
    elif guess > number:
        print("Your guess is too high")
    else:
        print("Your guess is too low")
```

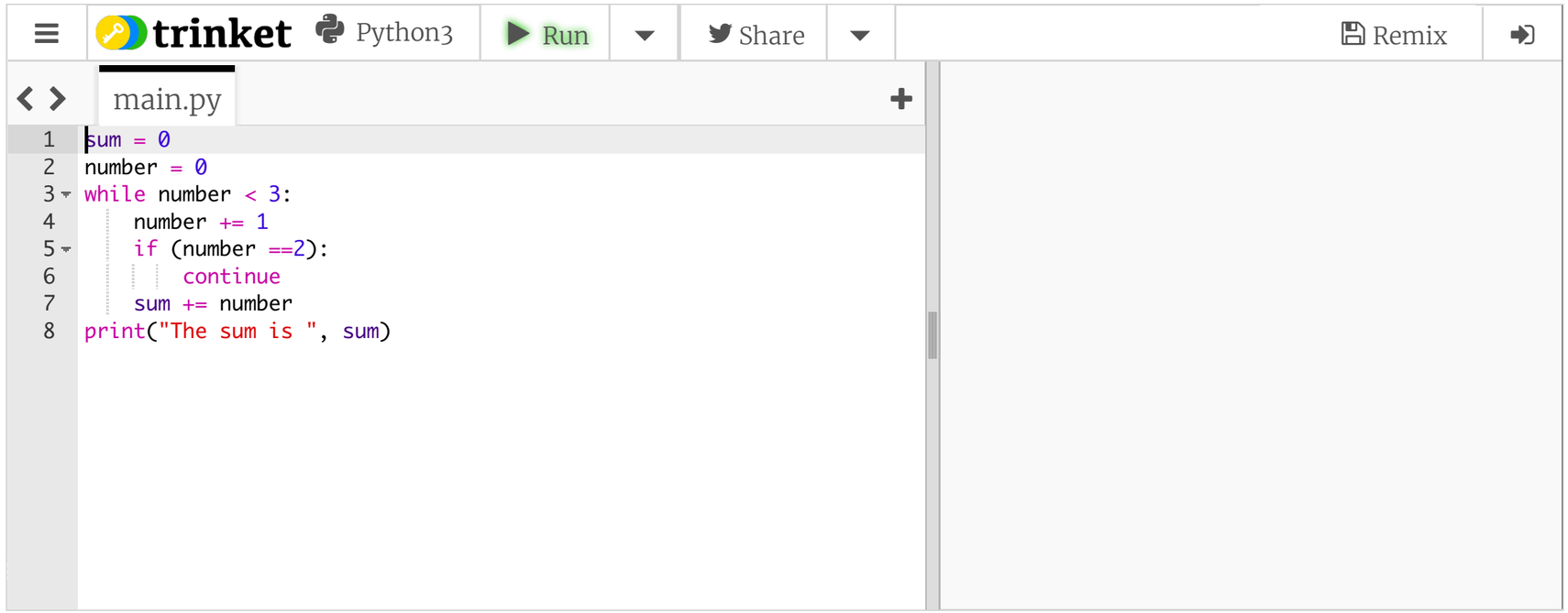
The keyword break

Instead of a condition, you can also use the keyword **break** to end the iteration of a loop.

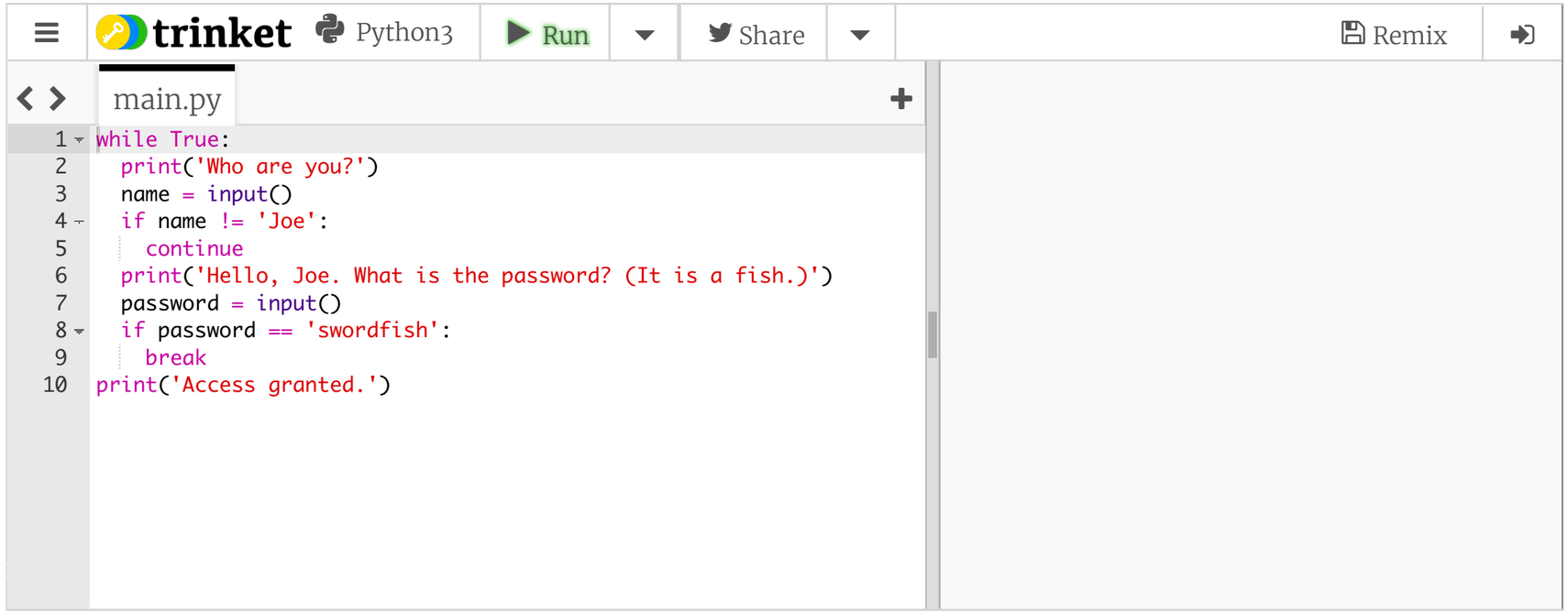


The keyword continue

You can use the keyword **continue** to ignore the remaining code in the iteration and jump to the next iteration



Combining break and continue



Exercise: quit the program with **Q**

Enable the use to enter some text and only quit the program if he clicks on "q" or "Q"

- Hint
- Show solution

Sentinel value

This is what you have just used in the previous exercise.

A sentinel value is a value entered by the user (with input) that will make the program stopped. You can put a sentinel value in your loop to decide when you want to **break** it, to stop it.

Exercise: compute average

Count positive and negative numbers and compute the average of numbers

Write a program that reads an unspecified number of integers, determines how many positive and negative values have been read, and computes the total and average of the input values (not counting zeros). Your program ends with the input 0. Display the average as a floating point number. Here is a sample run:

```
Enter an integer, the input ends if it is 0: 1
Enter an integer, the input ends if it is 0: 2
Enter an integer, the input ends if it is 0: -1
Enter an integer, the input ends if it is 0: 3
Enter an integer, the input ends if it is 0: 0
The number of positives is 3
The number of negatives is 1
The total is 5
The average is 1.25
Enter an integer, the input ends if it is 0: 0
You didn't enter any number
```

CITY UNIVERSITY LONDON

Solution: compute average

- Show solution

Structure of the for loop

```
for element in sequence:
    # statement
```

Where:

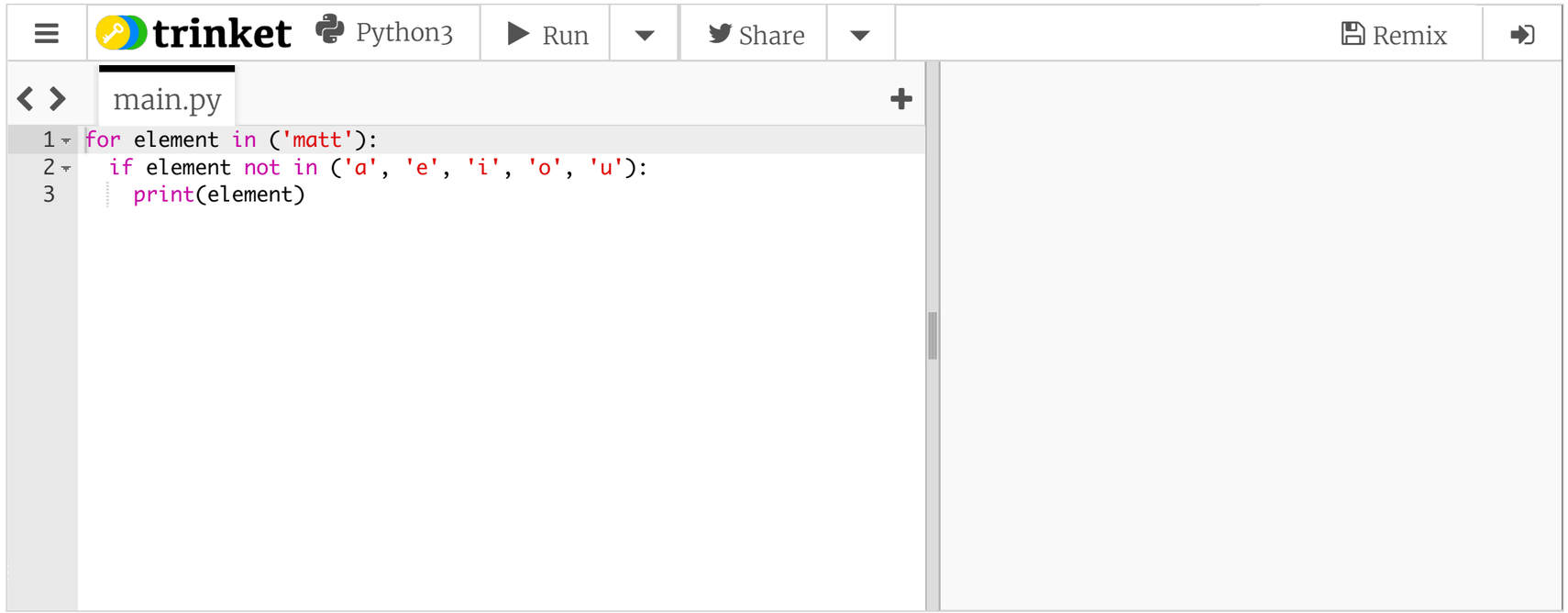
- **element** is a variable that is going to take the value of each element of the sequence
- **element** is NOT a keyword, it is a variable name, so you can give it whatever name you want
- the keywords are **for** and **in**
- notice the indentation that indicate the **body** of the loop (same as for **while**)

CITY UNIVERSITY LONDON

Example: a string is a sequence

A string is a sequence of characters on which we can iterate.

The value of **element** is going to be the value of each character of the string (each letter of the word) successively



The function range

You can create a sequence of number with the function **range()**

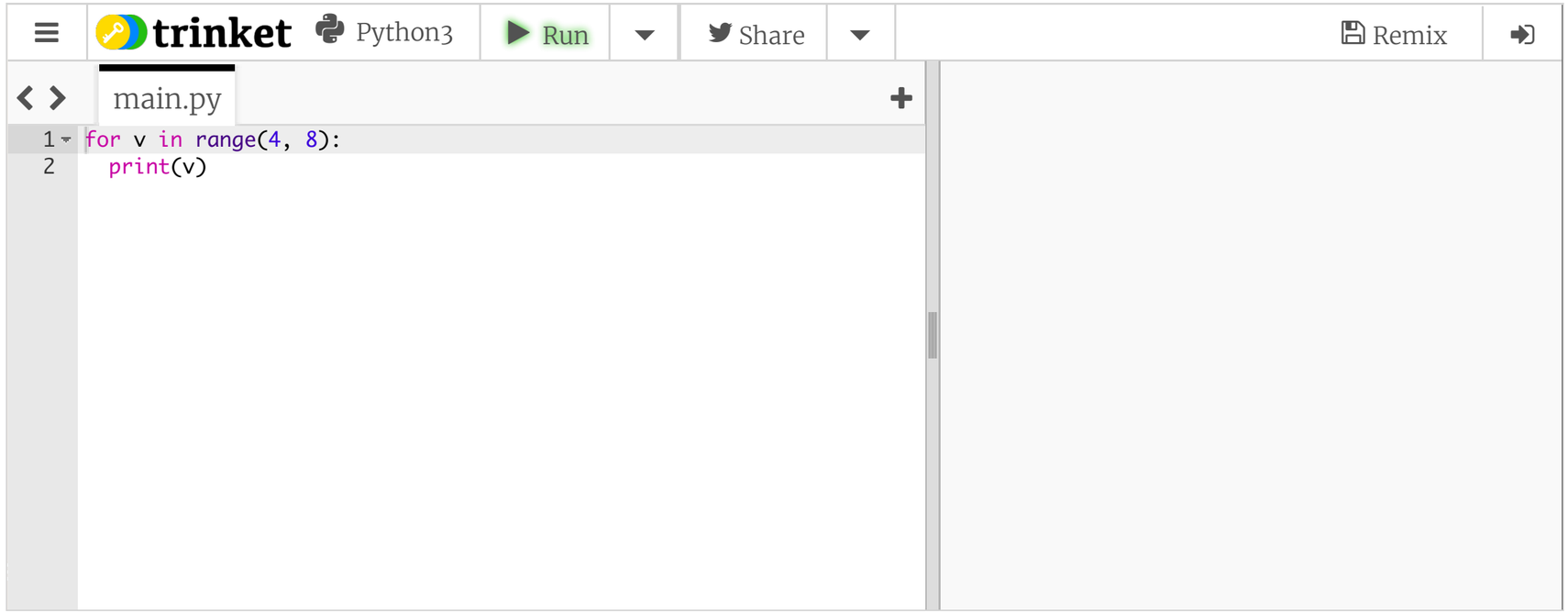
```
for element in range(initialValue, endValue, step):
    # statement
```

Where:

- **initialValue** and step value are optional arguments
- The default initialValue is 0 and the **endValue** is excluded
- **step** represents the increment and can be positive or negatove

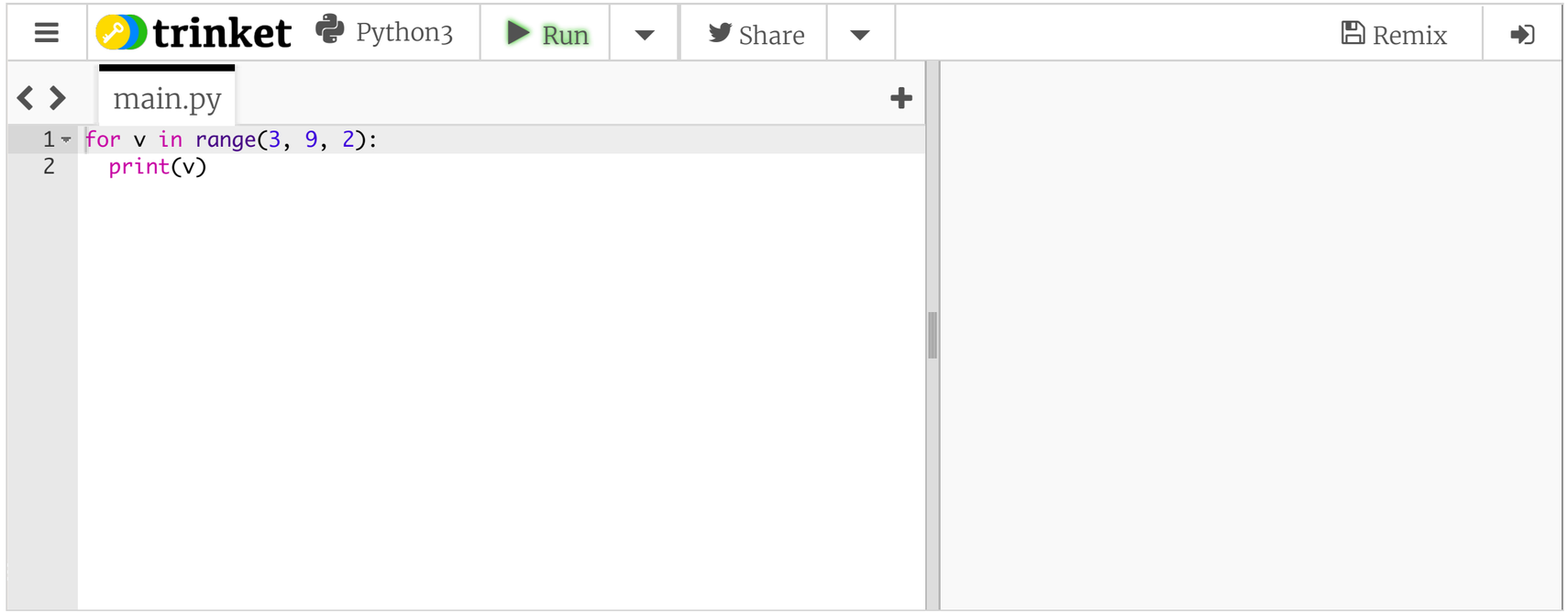
Example: range(initialValue, endValue)

Notice how the endValue is excluded



Example: range(initialValue, endValue, step)

Step specifies the increment



Exercise: conversion from miles to kilometer

Write a program that displays the following table (note that 1 mile is 1.609 kilometres):

```
Miles Kilometres
1 1.609
2 3.218
...
9 15.481
10 16.090
```

Solution: conversion from miles to kilometer

- Show solution

Exercise: Display leap years

Write a program that displays, ten per line, all the leap years in the twenty-first century (from year 2001 to 2100). The years are separated by exactly one space.

CITY UNIVERSITY LONDON

Solution: Display leap years

- Show solution

Sequence: check point

- Sequence are objects on which we can **iterate** (by using a while or a for loop)
- For each element you have one iteration
- At this point we have seen two types of sequence:
  - string: sequence of characters (letters)
  - range object: sequence of integer (numbers)
- NB: sequence are containers, they contain objects

Sequence: what is next?

- You will find and use sequences a lot in python
- We will see other built in python sequences
  - List
  - Tuple
  - Set
- Note that when we read a file in python, we also use iteration where each element of the loop is a line. We will reuse that when dealing with files.