Introduction to programming with Python

Session 3

Objectives

- Looping with while
- Looping with for

Motivation

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

- If the password was correct, we printed "Access Granted"
- If it was incorrect, 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. Even if the password was incorrect, the program would stop after only one try.

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

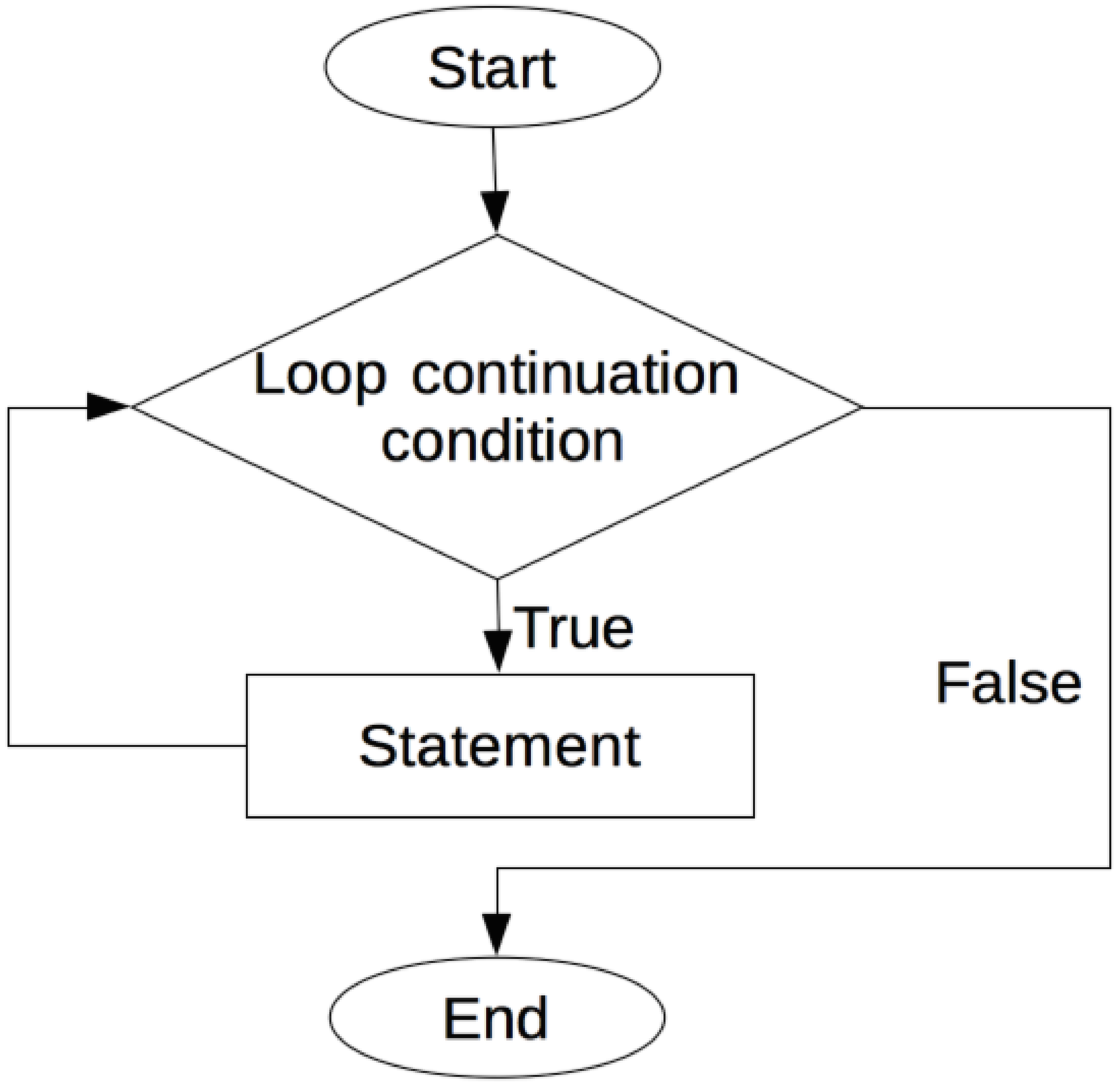
Structure of the while loop

```
while condition:
    # statement
```

Where:

- The **condition** is an expression that takes 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 the body of the loop is executed is an **iteration**

Structure of the while loop, flow chart



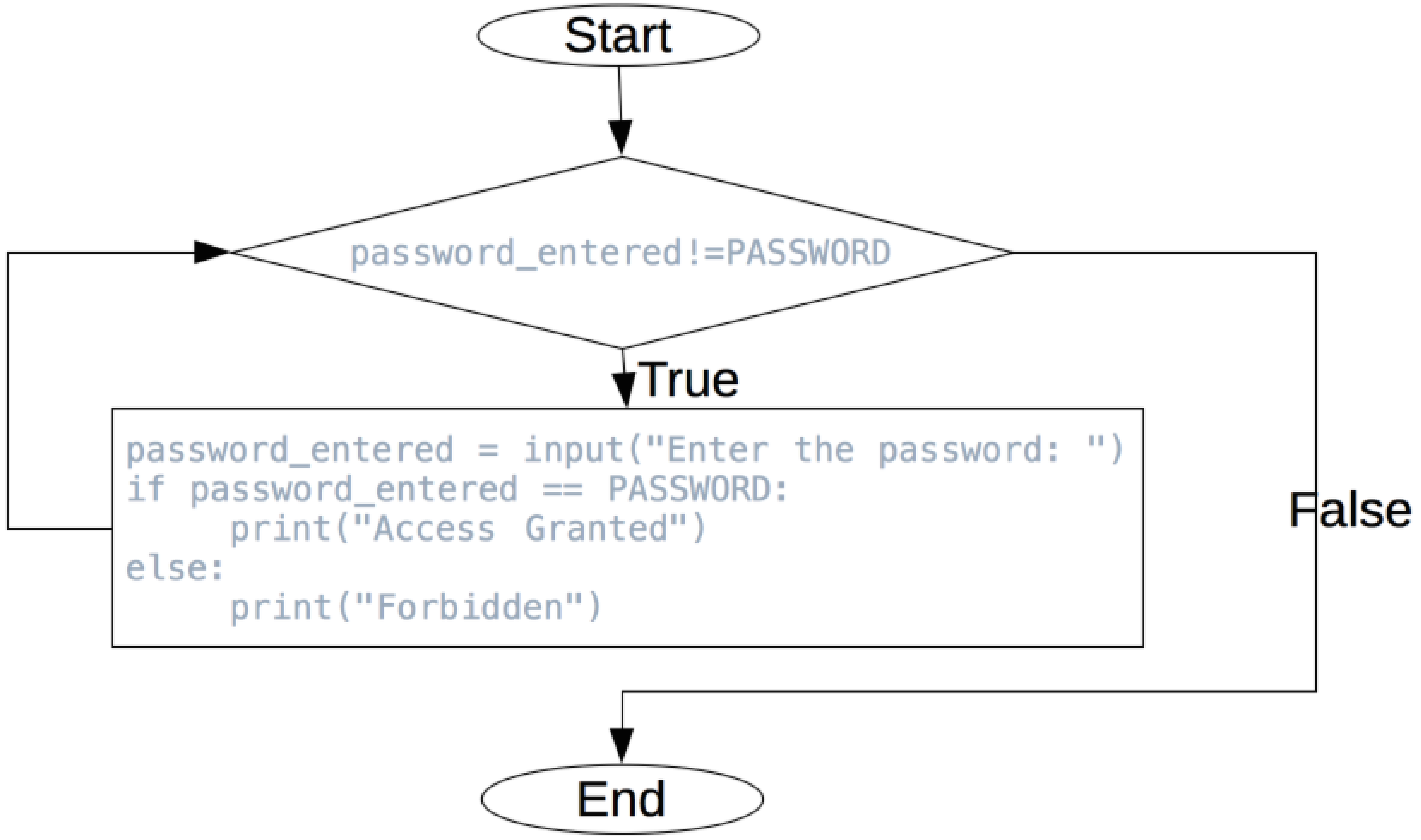
The while loop applied to our problem

```
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



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

Reminder: using variables

You cannot use 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 variables

You need to declare the variable *password_entered* before using it. If you don't you get an error:

```
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")
```

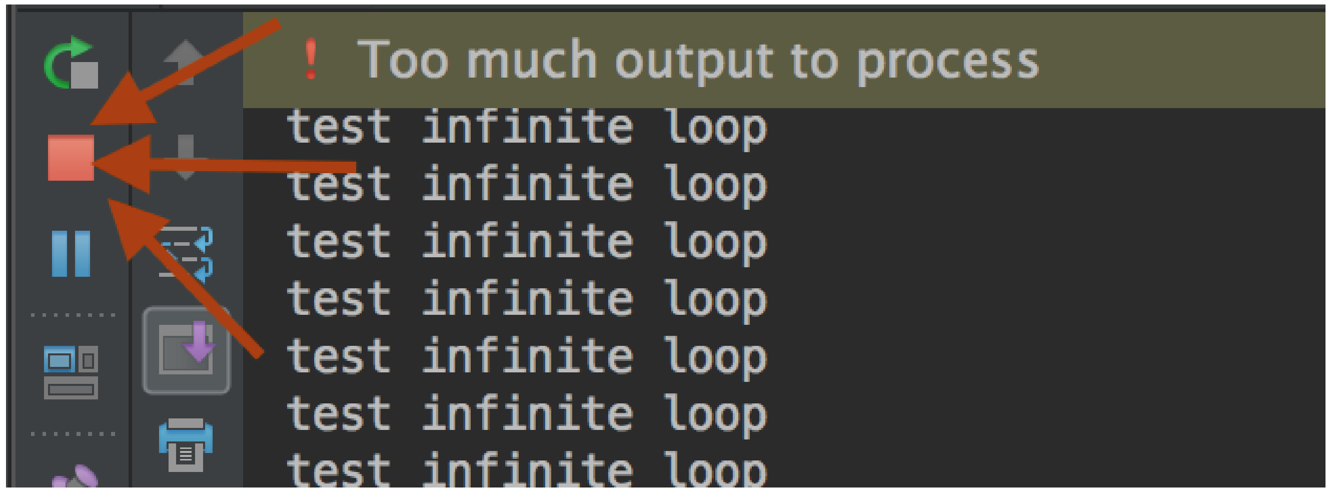
Reminder: conditions

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.** An infinite loop is always a bug!

```
while True: #do NOT do this! This is an example of what NOT to do.
    print('test infinite loop')
```

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



! Too much output to process

```
test infinite loop
test infinite loop
test infinite loop
test infinite loop
test infinite loop
test infinite loop
test infinite loop
```

How to avoid an infinite loop

Make sure that the condition changes to False at some point during the execution of the loop.

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 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 she 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 tries a number and presses enter, the program will say if the number is too high, too low or correct

Solution: Guess Number

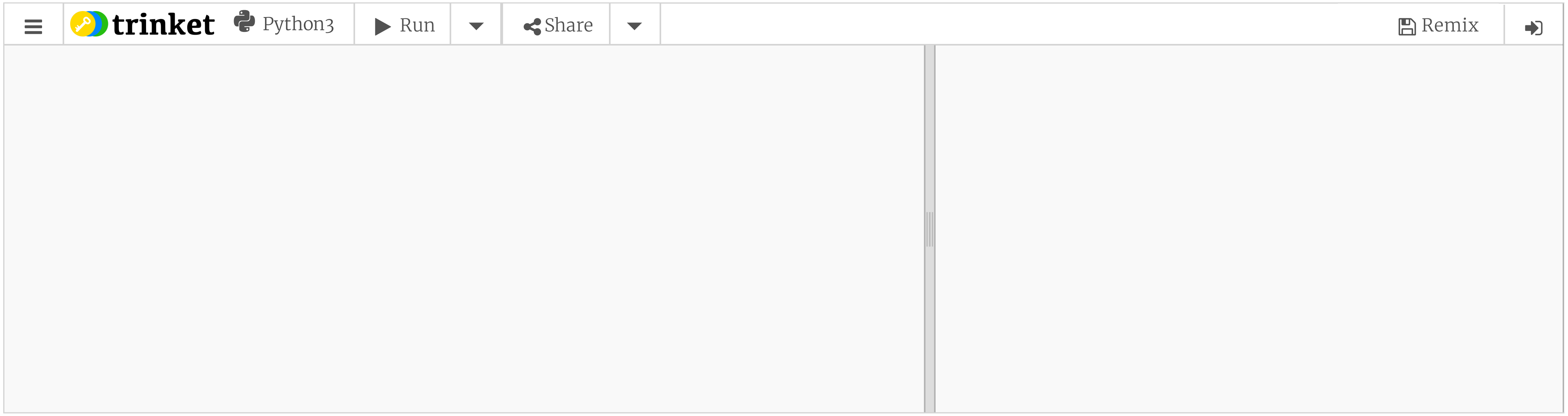
```
import random

# 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

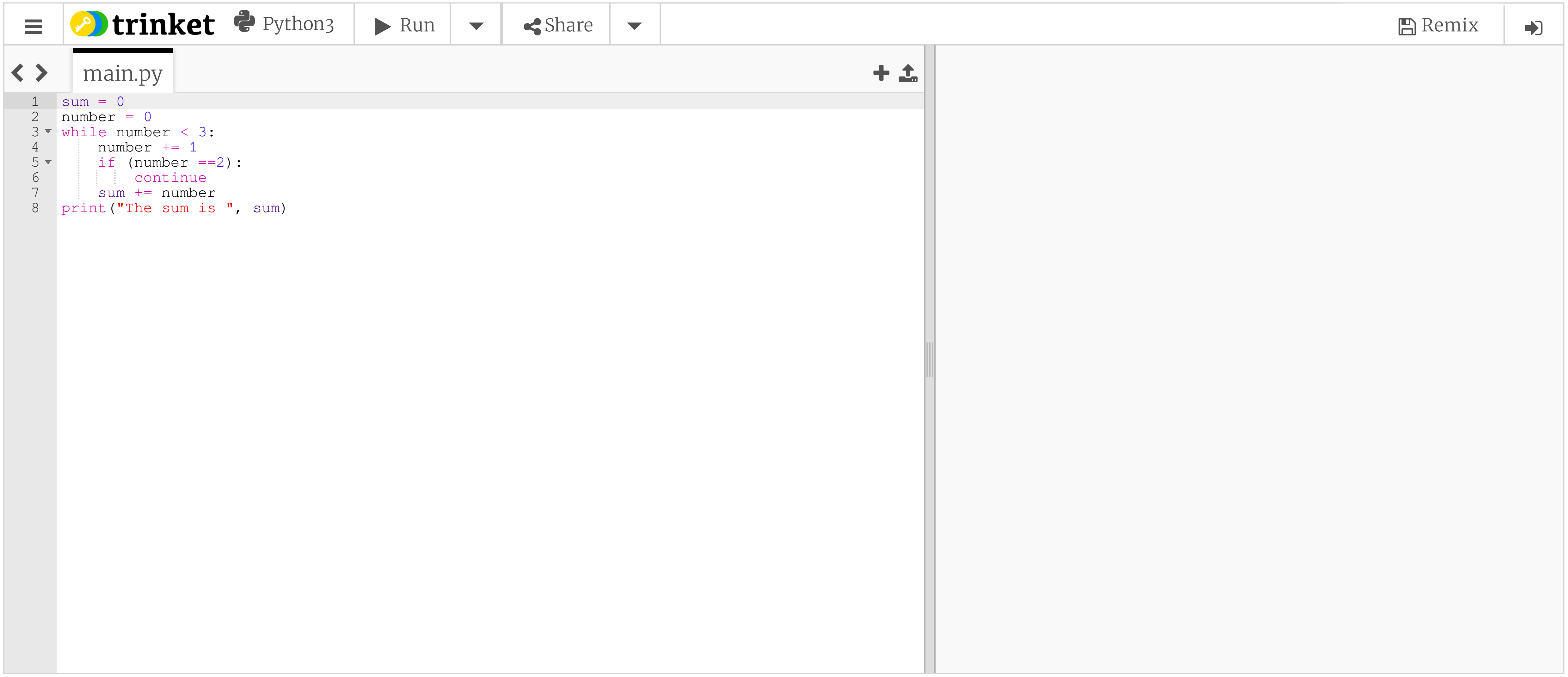
Instead of a condition, you can also use the keyword **break** to end the iteration of a loop.

Please note, this is not recommended. It is only presented here, because it has been used in a lot of legacy code. Remember, using an endless loop is **always** a bug.



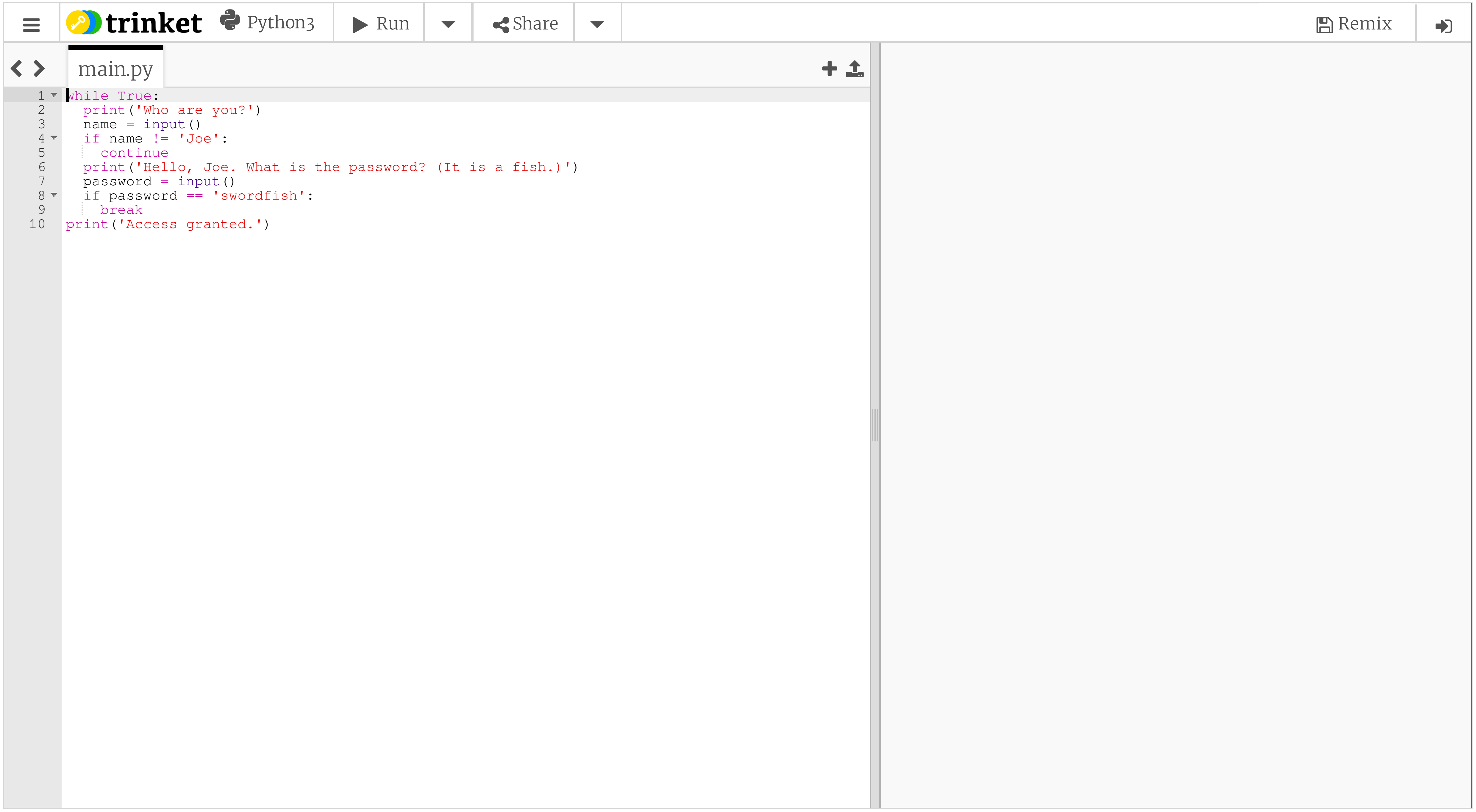
The keyword continue

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



```python
sum = 0
number = 0
while number < 3:
    number += 1
    if (number ==2):
        continue
    sum += number
print("The sum is ", sum)
```

Combining break and continue



```python
while True:
  print('Who are you?')
  name = input()
  if name != 'Joe':
    continue
  print('Hello, Joe. What is the password? (It is a fish.)')
  password = input()
  if password == 'swordfish':
    break
print('Access granted.')
```

Using break and continue today

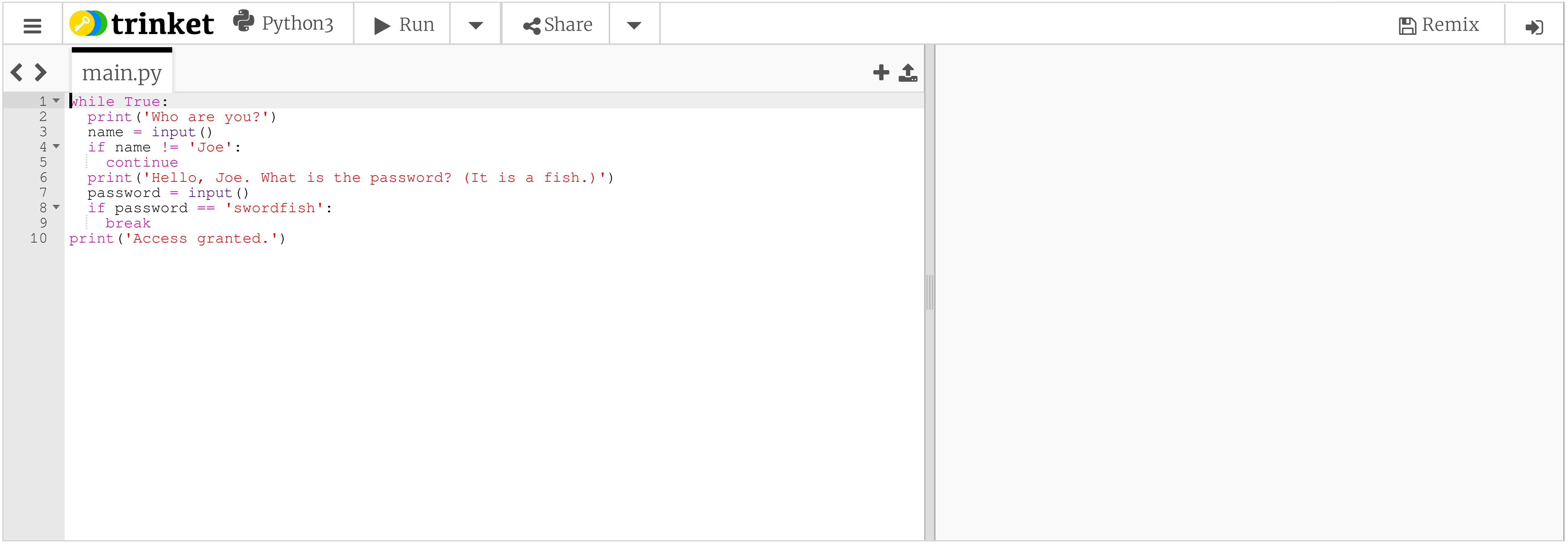
Both *break* and *continue* come from the distant past of computing. They are constructs of non-structured programs; non-structured programming was already considered problematic in 1968!

Understandably we should be avoiding the use of break and continue today (or simply leave it to the almost retiring software developers who studied how to code in the 60s!).

Today we should be using structured programming structures such as **if** blocks.

Using `if` instead of `break` and `continue`

Can you think how you can rewrite the code below using **if** blocks instead of break and continue?



```python
while True:
  print('Who are you?')
  name = input()
  if name != 'Joe':
    continue
  print('Hello, Joe. What is the password? (It is a fish.)')
  password = input()
  if password == 'swordfish':
    break
print('Access granted.')
```

Solution: Using `if` instead of `break` and `continue`

```
userIsNotAuthenticated = True
while userIsNotAuthenticated:
  print('Who are you?')
  name = input()
  if name == 'Joe':
    print('Hello, Joe. What is the password? (It is a fish.)')
    password = input()
    if password == 'swordfish':
      userIsNotAuthenticated = False
print('Access granted.')
```

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"

Can you think of two different solutions? One should use the `break` keyword, the other should only use `if`. Which one is better and why?

Solution

```
while True:
    my_input = input('Type "q" or "Q" to quit: ')
    if my_input.upper() == "Q":
        break
```

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 stop. 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
```

Solution: compute average

Solution

Download solution here:count_positive_negative_num.py (right click and save as)

Structure of the for loop

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

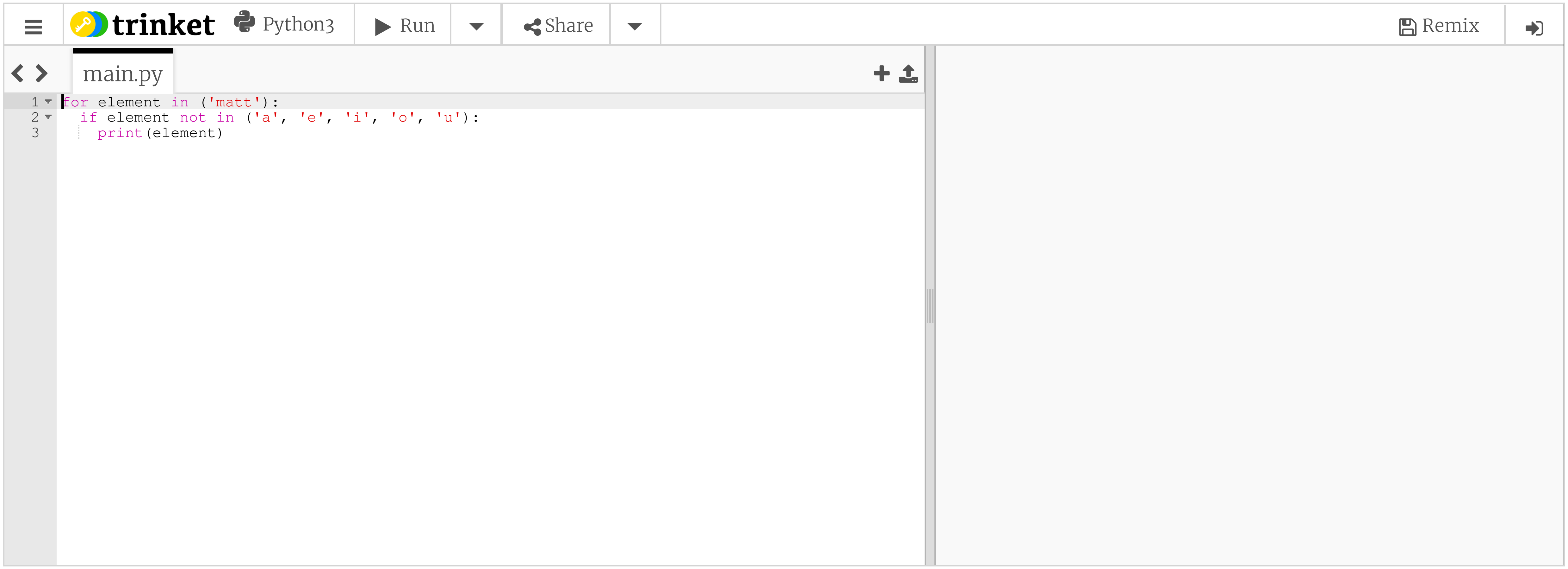
Where:

- **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 indicates the **body** of the loop (same as for **while**)

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



```python
for element in ('matt'):
  if element not in ('a', 'e', 'i', 'o', 'u'):
    print(element)
```

The function range

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

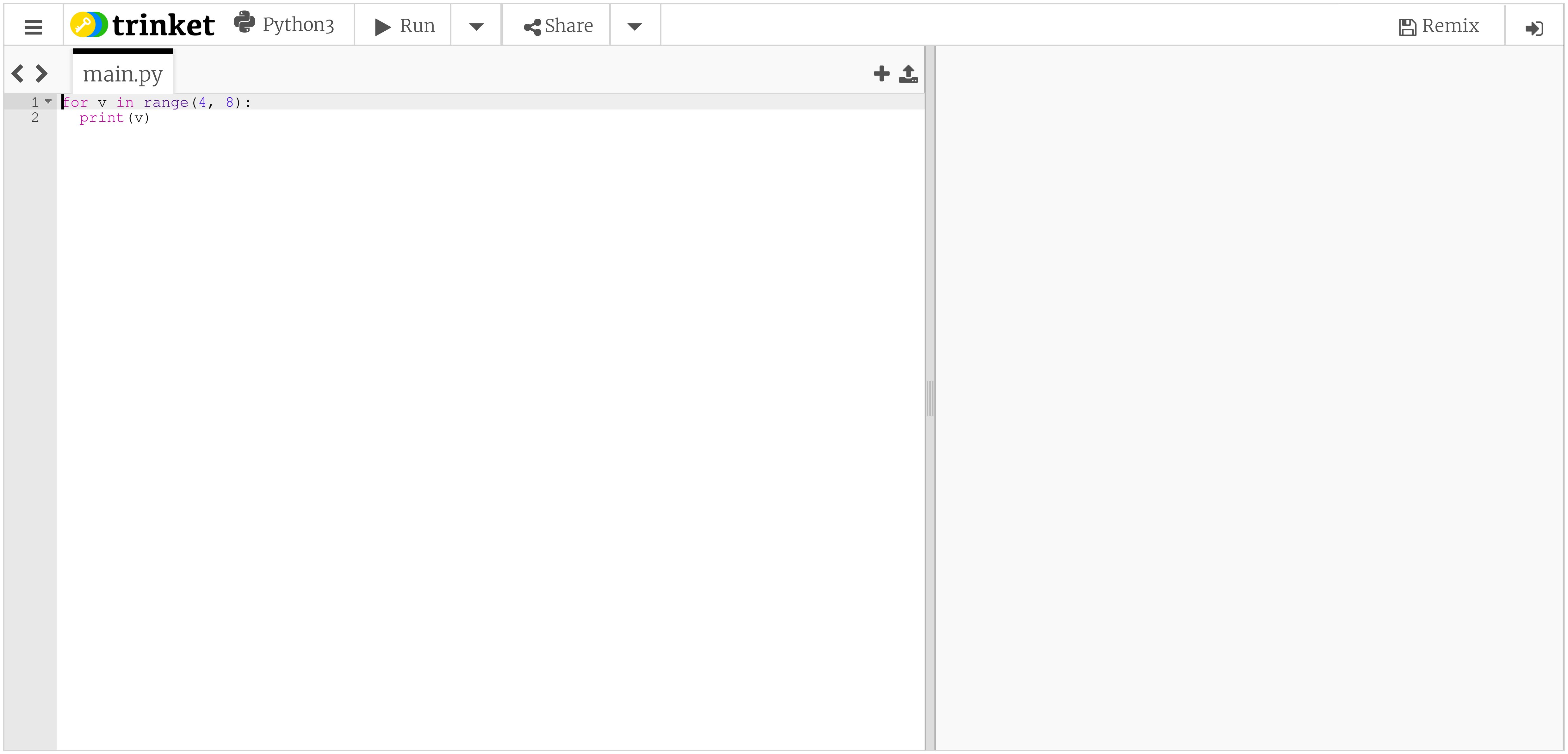
```
for element in range(initialValue, endValue, step):
    # statement
```

Where:

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

Example: range(initialValue, endValue)

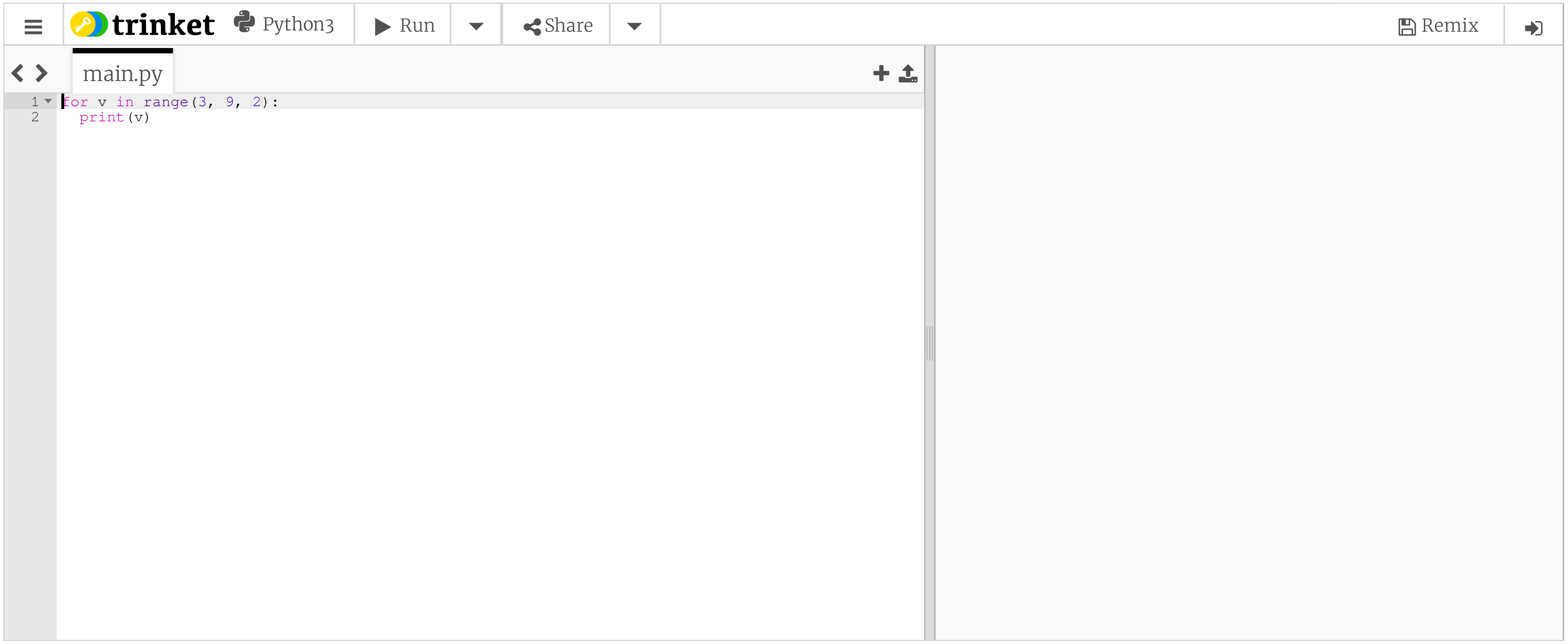
Notice how the endValue is excluded



```
for v in range(4, 8):
  print(v)
```

Example: range(initialValue, endValue, step)

Step specifies the increment



```
for v in range(3, 9, 2):
  print(v)
```

Exercise: conversion from miles to kilometres

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 kilometres

Solution

```
print("Miles"+ "   " + "Kilometres")
miles = 1
while miles <= 10:
    print(str(miles) + "   "+  str(miles * 1.609))
    miles += 1
```

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.

Solution: Display leap years

Solution

```
count = 1
for year in range(2001, 2100 + 1):
    if year % 400 == 0 or \
        (year % 4 == 0 and year % 100 != 0):
        if count % 10 == 0:
            print(year)
        else:
            print(year, end = " ")

        count += 1
```

Exercise Bonus: find if a string is a Palindrome

A string is a palindrome if it is identical forward and backward. For example “anna”, “civic”, “level” and “hannah” are all examples of palindromic words. Write a program that reads a string from the user and uses a loop to determine whether or not it is a palindrome. Display the result, including a meaningful output message.

Solution: Palindrome

Solution

```
line = input('Enter a string: ')

is_palindrome = True

for i in range(0, len(line) // 2):
    if line[i] != line[len(line) - 1 - i]:
        is_palindrome = False
        break

if is_palindrome:
    print(line, "is a palindrome")
else:
    print(line, "is not a palindrome")
```

Bonus Exercise: Caesar Cipher

The idea behind this cipher is simple. Each letter in the original message is shifted by 3 places. As a result, A becomes D, B becomes E, C becomes F, D becomes G, etc. The last three letters in the alphabet are wrapped around to the beginning: X becomes A, Y becomes B and Z becomes C. Non-letter characters are not modified by the cipher.

Write a program that implements a Caesar cipher. Allow the user to supply the message and the shift amount, and then display the shifted message. Your program should also support negative shift values so that it can be used both to encode messages and decode messages.

Solution: Caesar Cipher, Ciphering

Solution

Download this file: caesar_cipher.py (right click and save as)

Solution: Caesar Cipher, Deciphering

Solution

Download this file: caesar_decipher.py (right click and save as)

Exercise: Refactoring Caesar Cipher

At this point we have one script for ciphering and an other for deciphering a message.

But we can notice that the scripts are very similar, and parts of the code are repeated.

We do not want that in programming, we want to try to respect the DRY principle (Don't Repeat Yourself)

Refactor the 2 previous scripts to make it one so that when we run it, we ask the user if he wants to cipher or decipher the entered message

Sequences: check point

- Sequences 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 sequences:
    - string: sequences of characters (letters)
    - range object: sequences of integer (numbers)
- NB: sequences are containers, they contain objects

Sequences: 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 revisit that when dealing with files.