



# Open Source Simulator Software

PROGRAMMING LANGUAGE SIMULATOR SOFTWARE  
ARTIFICIAL INTELLIGENCE ADD-ON USING THE SIMULATOR  
PRACTICE EXERCISES

*KA220VET Erasmus+ Projesi*

# Why Learn Python?

---

Beginner-  
friendly

Used  
worldwide

Future career  
language

# What is Python?

---

Python is a high-level programming language.

Easy to read and learn.

# What is Python?

---



It is easy to learn because it has a simple syntax.



It is similar to the English language.



It uses new lines to complete a command instead of semicolons and parentheses.



It uses indentation (space at the beginning of the line) to define the scope of loops, functions, and classes.



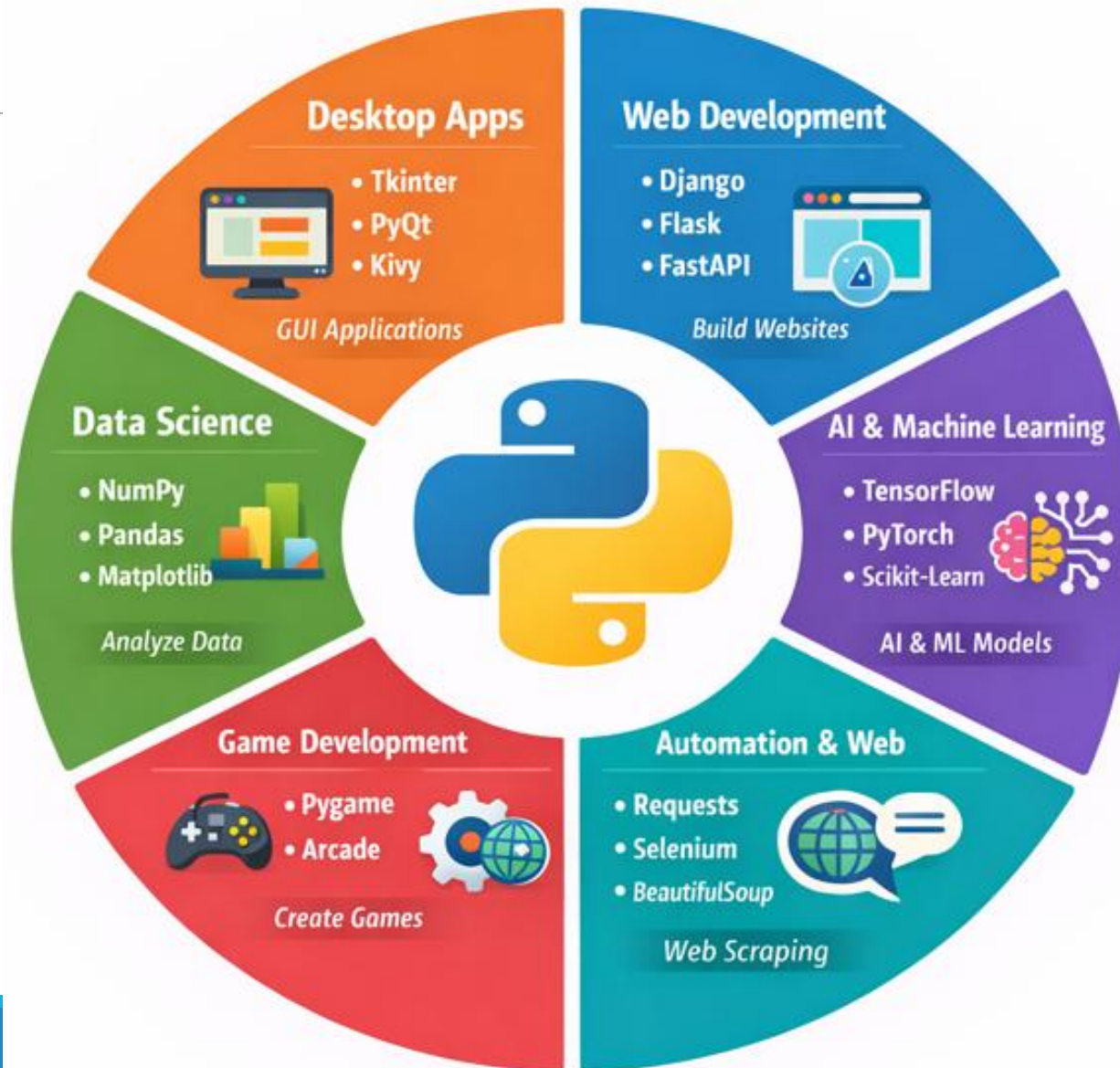
It can run on all platforms such as Windows, Linux, and Mac.

# Usage Areas:

---

- System programming
- User interface programming
- Web development
- Network programming
- Game development
- Data science, machine learning
- Data analysis
- Artificial Intelligence (AI)
- Scripting and tools

# Usage Areas & Libraries



# Python in Daily Life

---



SOCIAL MEDIA



SMART SYSTEMS



ONLINE SERVICES

How Will We  
Code?

---

We can use

**online-python.com**

---

No installation  
needed

# Python Installation

---

## For Windows:

- Go to the official Python website at <https://www.python.org/>.
- Click on the **Downloads** menu. Download the latest version of Python (e.g., Python 3.14.0).
- Run and Setup program

# Editör

---

➤ Notebook, Notepad++

➤ Visual Studio Code (<https://code.visualstudio.com/>)

# Running the First Code

---

```
print("Hello World")
```

# Variables

---

Variables store data in programs

A variable is created the moment you first assign a value to it.

**Codes:**

```
x = 5  
y = "John"  
print(x)  
print(y)
```

**Output**



```
5  
John
```

## Variable Example

Variables do not need to be declared with any particular *type*, and can even change type after they have been set.

```
name = "Alex " #name is of type str
```

```
age = 16        # age is of type int
```

## Variable Names

### Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alphanumeric characters and underscores (A-z, 0-9, and \_ )
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- A variable name cannot be any of the Python keywords.

# Variable Names

---

## Example

Legal variable names:

```
myvar = "John"  
my_var = "John"  
_my_var = "John"  
myVar = "John"  
MYVAR = "John"  
myvar2 = "John"
```

## Example

Illegal variable names:

```
2myvar = "John"  
my-var = "John"  
my var = "John"
```

# Python Data Types

---

- **string** (textual),
- **numbers** (numeric),
- **boolean**,
- **list**,
- **tuple**,
- **Dictionary**
- **set**

# String (Textual) Data Type

---

String data types are used to store textual expressions.

Values are written inside single quotes ( ' ') or double quotes ( " " ).

The characters can be letters (a,..z ), numbers (1, 9, 2, 3), or special symbols (&,/).

## Examples

name="Mert"

surname='Yilmaz'

# Numeric Data Type

---

Used to store numeric values. There are three main numeric types:

**int (Integer):** Used for whole numbers (e.g., 5, -10, 100).

**float (Floating Point):** Used for decimal numbers (e.g., 3.14, -0.5).

**complex:** Used for complex numbers (e.g.,  $3+5j$ ).

# Python Lists

---

Lists are used to store multiple items in a single variable.

## Examples:

```
fruits = ["apple", "banana", "cherry"]
```

```
sayı=[10,20,30,40,50,60,70]
```

```
sehir=["İzmir", "İstanbul", "Ankara", "Bolu"]
```

```
first_list=["Ankara", 312, 0.6]
```

# Input and Output

---



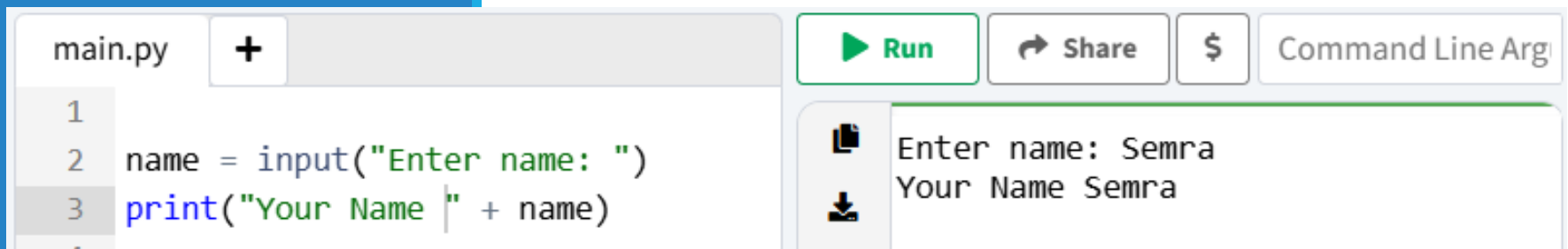
INPUT() GETS DATA



PRINT() SHOWS OUTPUT

# Input Variables

- We use the `input()` command to get a value from the user.
- We assign the received value to a variable.



The screenshot shows a Python IDE interface. On the left, a file named 'main.py' is open, containing the following code:

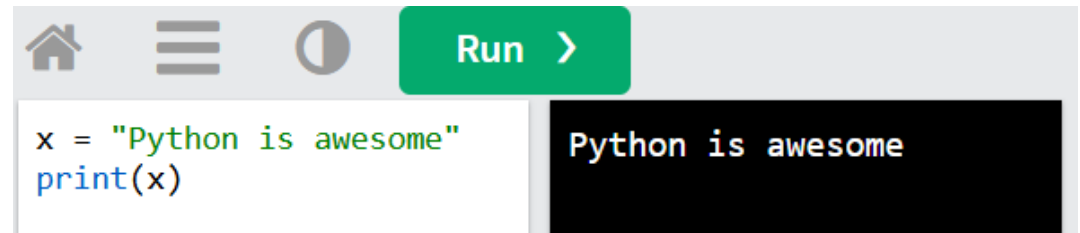
```
1  
2 name = input("Enter name: ")  
3 print("Your Name " + name)
```

On the right, there are buttons for 'Run', 'Share', and 'Command Line Arg'. Below these buttons, the output of the program is displayed in a terminal window:

```
Enter name: Semra  
Your Name Semra
```

# Output Variables

The `print()` function is often used to output variables.

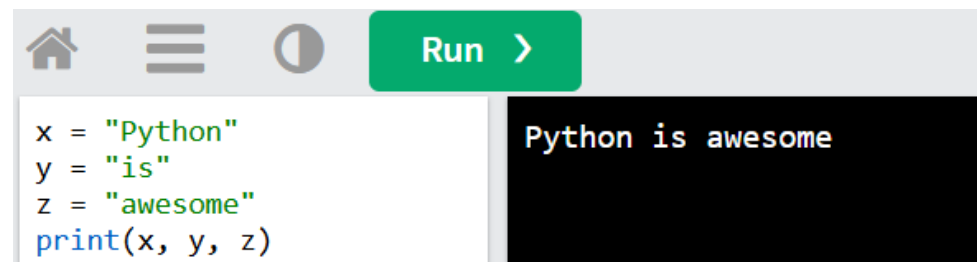


A code editor interface with a toolbar at the top containing a home icon, a menu icon, a moon icon, and a green 'Run >' button. The code area contains two lines: `x = "Python is awesome"` and `print(x)`. To the right of the code area is a black output console displaying the text `Python is awesome` in white.

```
x = "Python is awesome"
print(x)
```

Python is awesome

In the `print()` function, you output multiple variables, separated by a comma:

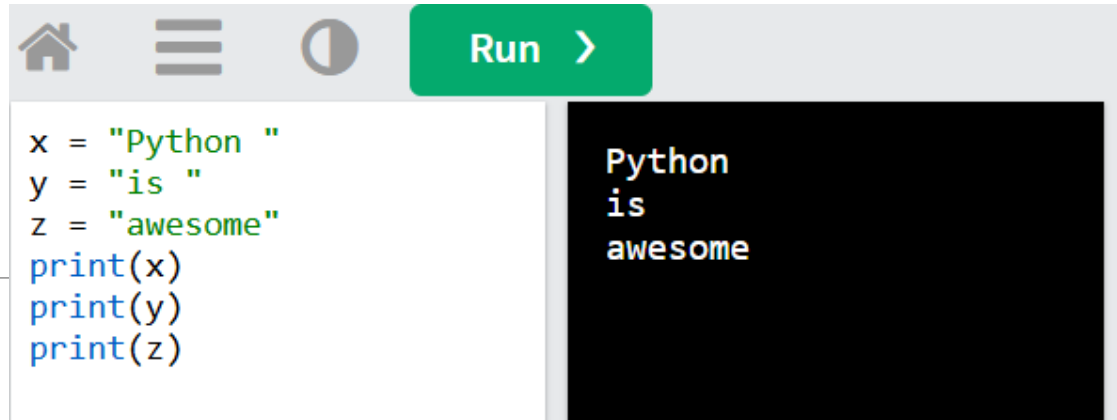


A code editor interface with a toolbar at the top containing a home icon, a menu icon, a moon icon, and a green 'Run >' button. The code area contains four lines: `x = "Python"`, `y = "is"`, `z = "awesome"`, and `print(x, y, z)`. To the right of the code area is a black output console displaying the text `Python is awesome` in white.

```
x = "Python"
y = "is"
z = "awesome"
print(x, y, z)
```

Python is awesome

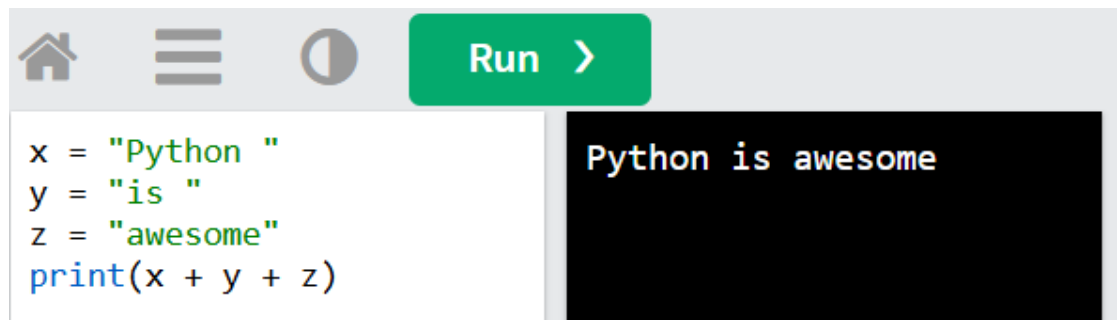
You can also use the + operator to output multiple variables:



The image shows a Python IDE window with a light gray header bar containing a home icon, a menu icon, a moon icon, and a green 'Run >' button. The main area is split into two panes. The left pane contains the following Python code:

```
x = "Python "  
y = "is "  
z = "awesome"  
print(x)  
print(y)  
print(z)
```

The right pane shows the output of the code, which is the text 'Python is awesome' displayed on three separate lines.



The image shows a Python IDE window with a light gray header bar containing a home icon, a menu icon, a moon icon, and a green 'Run >' button. The main area is split into two panes. The left pane contains the following Python code:

```
x = "Python "  
y = "is "  
z = "awesome"  
print(x + y + z)
```

The right pane shows the output of the code, which is the text 'Python is awesome' displayed on a single line.

# If Statement

---

Used for decision making

An "if statement" is written by using the if keyword.

# How If Statements Work

---

The if statement evaluates a condition (an expression that results in True or False).

If the condition is true, the code block inside the if statement is executed.

If the condition is false, the code block is skipped.

if (condition):

..... True code

continue here

# If Example

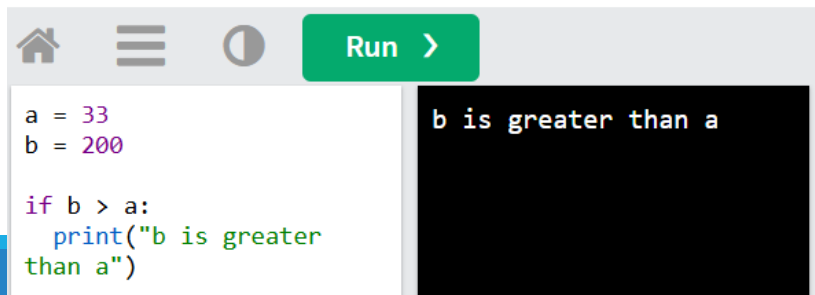
## Example

If statement:

```
a = 33
b = 200
if b > a:
    print("b is greater than a")
```

In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a.

As a is 33, and b is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

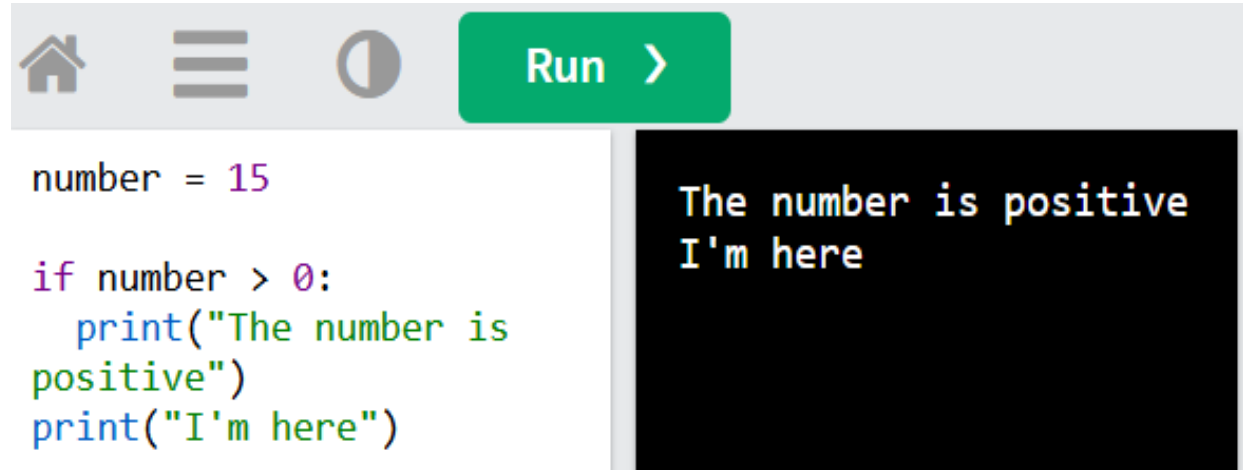
A screenshot of a code editor interface. The editor has a light gray header with icons for home, menu, and a circular icon, and a green 'Run' button with a right arrow. The code area on the left shows the same Python code as the example: 

```
a = 33
b = 200
if b > a:
    print("b is greater than a")
```

 The output area on the right, which has a black background, displays the text `b is greater than a` in white.

# If Example

- For number= 15;



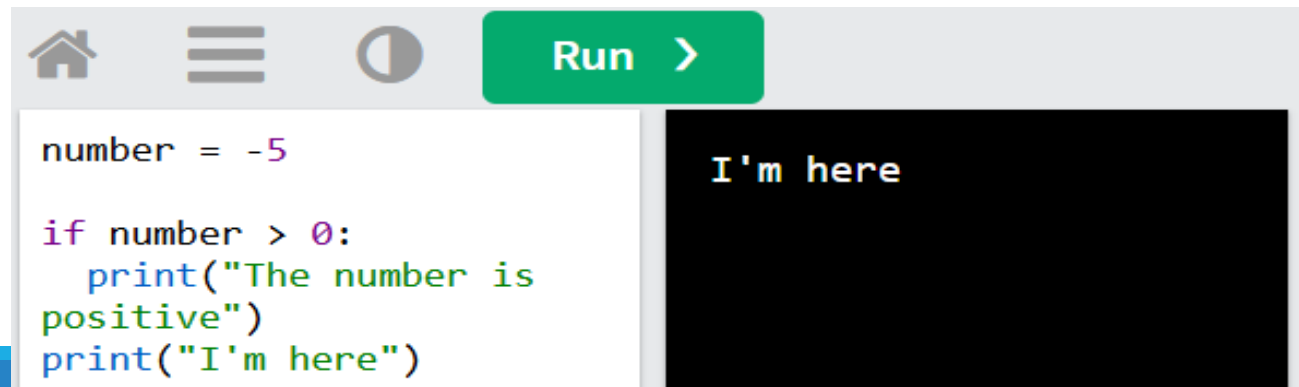
The image shows a Python IDE interface. At the top, there is a toolbar with a home icon, a menu icon, a play icon, and a green 'Run >' button. Below the toolbar, the code editor contains the following Python code:

```
number = 15

if number > 0:
    print("The number is positive")
    print("I'm here")
```

To the right of the code editor is a black output window with white text that reads: "The number is positive" followed by "I'm here" on the next line.

- For number= -5;



The image shows a Python IDE interface. At the top, there is a toolbar with a home icon, a menu icon, a play icon, and a green 'Run >' button. Below the toolbar, the code editor contains the following Python code:

```
number = -5

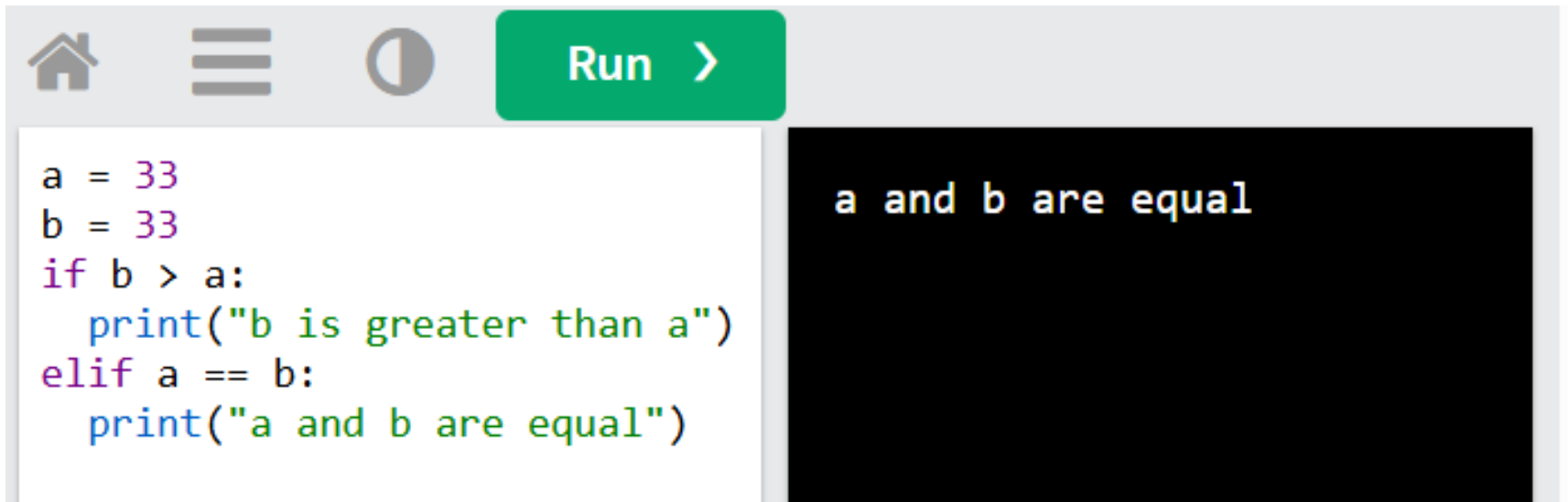
if number > 0:
    print("The number is positive")
    print("I'm here")
```

To the right of the code editor is a black output window with white text that reads: "I'm here".

# Python Elif Statement

The elif keyword is Python's way of saying "if the previous conditions were not true, then try this condition"

## Example

A screenshot of a Python IDE interface. At the top, there is a toolbar with a home icon, a menu icon, a circular icon, and a green 'Run' button with a right-pointing arrow. Below the toolbar, the code editor on the left contains the following Python code:

```
a = 33
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
```

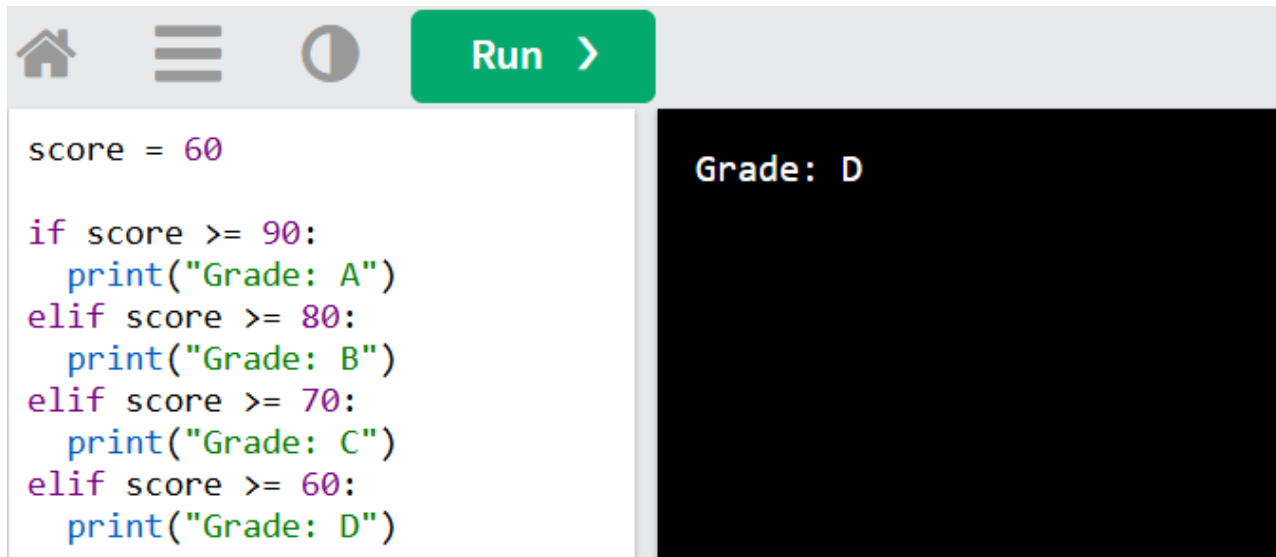
The output console on the right, which has a black background, displays the text 'a and b are equal' in a yellow monospace font.

# Multiple Elif Statements

---

You can have as many elif statements as you need.

Python will check each condition in order and execute the first one that is true.



The screenshot shows a Python IDE interface. At the top, there is a toolbar with icons for home, menu, and a circular icon, followed by a green 'Run' button with a right-pointing arrow. Below the toolbar, the code editor displays the following Python code:

```
score = 60

if score >= 90:
    print("Grade: A")
elif score >= 80:
    print("Grade: B")
elif score >= 70:
    print("Grade: C")
elif score >= 60:
    print("Grade: D")
```

To the right of the code editor is a black terminal window with the output 'Grade: D' displayed in white text.

# if example – Let's Try

ivebsim\_pythoncode >  if\_2.py > ...

```
1  score = 60
2
3  if score >= 85:
4      print("Your grade is: 5")
5
6  elif score >= 70:
7      print("Your grade is: 4")
8
9  elif score >= 55:
10     print("Your grade is: 3")
11
12 else:
13     print("You failed the exam.")
14
```

## Python Operators

The most used operations are

Arithmetic operators

Comparison operators

Logical operators

# Python Operators



Operators are used to perform operations on variables and values.



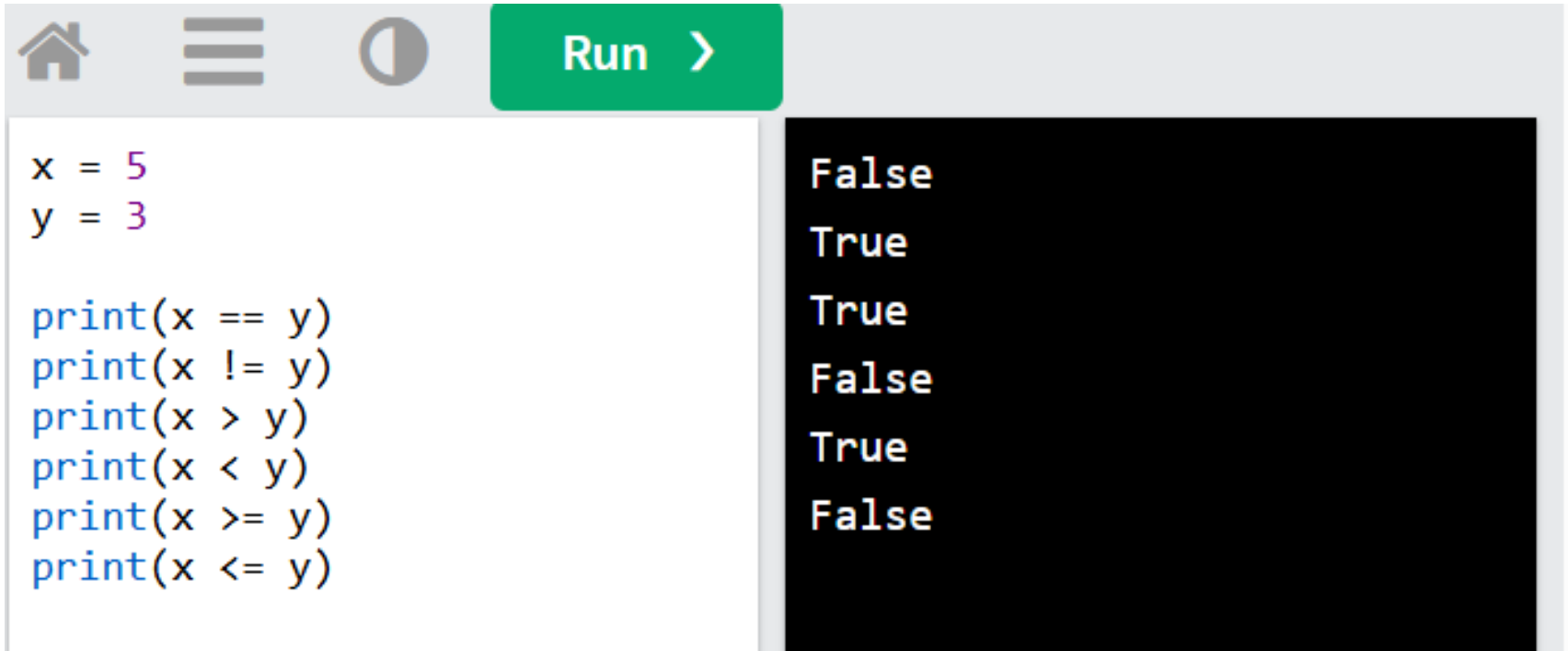
Operators are used in decision making and used in Loops

# Comparison operators

Comparison operators are used to compare two values:

| Operator | Name                     | Example |
|----------|--------------------------|---------|
| ==       | Equal                    | x == y  |
| !=       | Not equal                | x != y  |
| >        | Greater than             | x > y   |
| <        | Less than                | x < y   |
| >=       | Greater than or equal to | x >= y  |
| <=       | Less than or equal to    | x <= y  |

# Comparison operators - Examples

A screenshot of a Python IDE interface. At the top, there is a toolbar with icons for home, menu, and a circular icon, followed by a green 'Run' button with a right-pointing arrow. Below the toolbar, the left pane contains Python code: 'x = 5', 'y = 3', and six 'print' statements using comparison operators. The right pane shows the output of these statements: 'False', 'True', 'True', 'False', 'True', and 'False'.

```
x = 5
y = 3

print(x == y)
print(x != y)
print(x > y)
print(x < y)
print(x >= y)
print(x <= y)
```

False  
True  
True  
False  
True  
False

# Logical operators

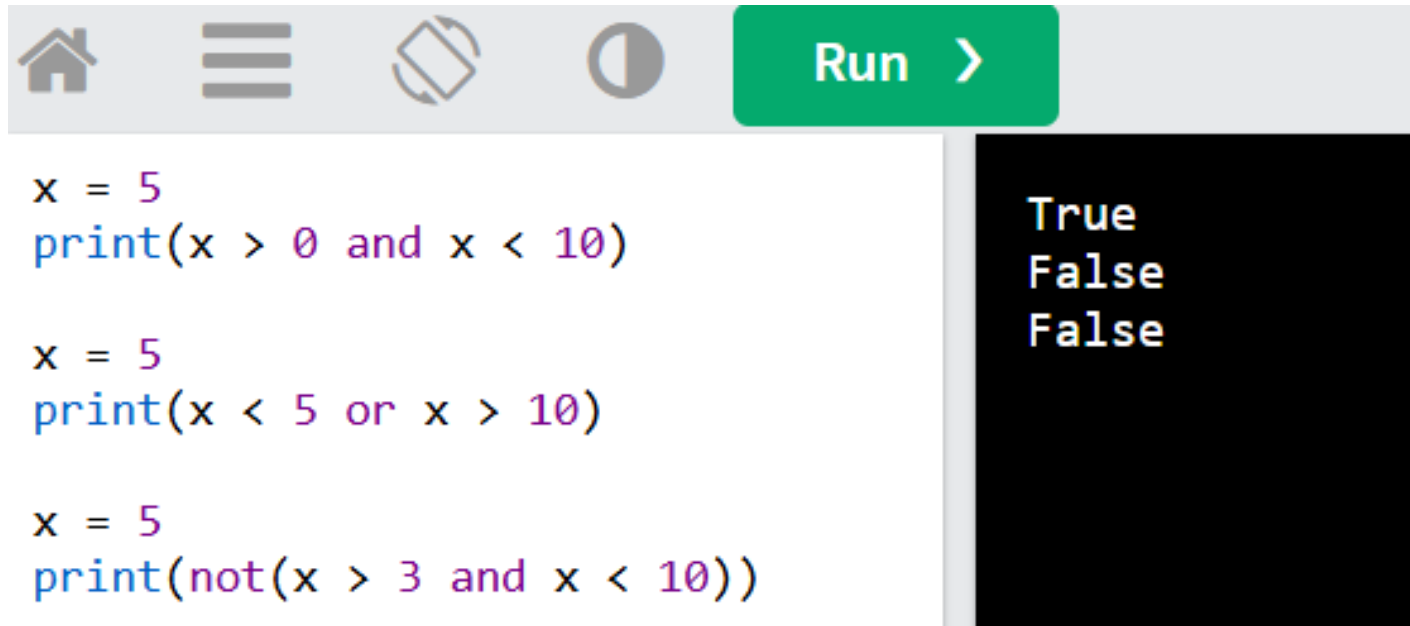
---

Logical operators are used to compare two values:

We typically use these within decision structures and loops.

| Operator | Description                                             | Example                                  |
|----------|---------------------------------------------------------|------------------------------------------|
| and      | Returns True if both statements are true                | <code>x &lt; 5 and x &lt; 10</code>      |
| or       | Returns True if one of the statements is true           | <code>x &lt; 5 or x &lt; 4</code>        |
| not      | Reverse the result, returns False if the result is true | <code>not(x &lt; 5 and x &lt; 10)</code> |

# Logical operators - Examples



The image shows a Python IDE interface. At the top, there is a toolbar with icons for home, menu, undo, and redo, followed by a green 'Run' button with a right arrow. Below the toolbar, the code editor contains three code blocks. The first block sets x to 5 and prints the result of x > 0 and x < 10. The second block sets x to 5 and prints the result of x < 5 or x > 10. The third block sets x to 5 and prints the result of not(x > 3 and x < 10). To the right of the code editor, a black output window displays the results of the three print statements: True, False, and False.

```
x = 5
print(x > 0 and x < 10)

x = 5
print(x < 5 or x > 10)

x = 5
print(not(x > 3 and x < 10))
```

True  
False  
False

# Arithmetic Operators

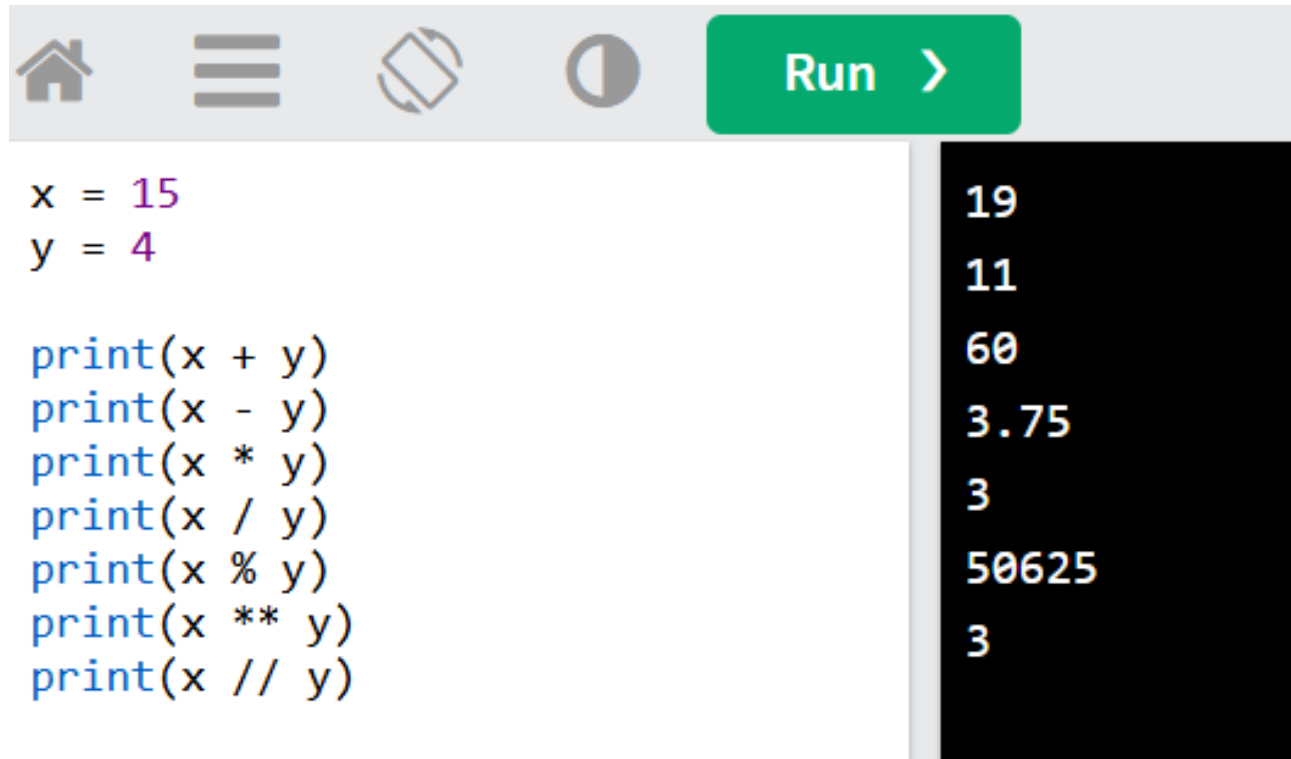
---

We use them in mathematical operations.

| Operator | Name           | Example  |
|----------|----------------|----------|
| +        | Addition       | $x + y$  |
| -        | Subtraction    | $x - y$  |
| *        | Multiplication | $x * y$  |
| /        | Division       | $x / y$  |
| %        | Modulus        | $x \% y$ |
| **       | Exponentiation | $x ** y$ |
| //       | Floor division | $x // y$ |

# Arithmetic Operators - Examples

---



The image shows a Python IDE interface. At the top, there is a toolbar with icons for home, menu, undo, and redo, followed by a green 'Run' button with a right-pointing arrow. Below the toolbar, the code editor contains the following Python code:

```
x = 15
y = 4

print(x + y)
print(x - y)
print(x * y)
print(x / y)
print(x % y)
print(x ** y)
print(x // y)
```

To the right of the code editor, a black output console displays the results of the print statements in yellow text:

```
19
11
60
3.75
3
50625
3
```

# LOOPS



In Python programming, we use **loops** to run a block of code **repeatedly without writing it again and again.**



Loops help us **save time**, make our code **shorter, cleaner, and easier to understand.**



They also allow us to easily work with multiple values such as lists and arrays.

# LOOPS

# For Loop

---



We use the **for loop** to repeat a block of code a **specific number of times** or **for each element in a list or sequence**.



If the number of repetitions is **known in advance**, the **for loop** is the best and simplest choice.

# Example: Counting from 1 to 15 using a for loop

ivebsim\_pythoncode >  for1.py

```
1  print(1)
2  print(2)
3  print(3)
4  print(4)
5  print(5)
6  print(6)
7  print(7)
8  print(8)
9  print(9)
10 print(10)
11 print(11)
12 print(12)
13 print(13)
14 print(14)
15 print(15)
```

```
for i in range(1,16):
    print(i)
```

# Using the range() Function with FOR:

---

## Syntax

The range() function can take up to three arguments:

- range(start, stop, step)

# Range Examples

---

| Code                         | Output        | Description                   |
|------------------------------|---------------|-------------------------------|
| <code>range(5)</code>        | 0, 1, 2, 3, 4 | Starts at 0, stops before 5.  |
| <code>range(2, 6)</code>     | 2, 3, 4, 5    | Starts at 2, stops before 6.  |
| <code>range(0, 10, 2)</code> | 0, 2, 4, 6, 8 | Starts at 0, increments by 2. |
| <code>range(5, 0, -1)</code> | 5, 4, 3, 2, 1 | Counts down from 5 to 1.      |

# Range Example- Lets' Try

---

main.py

+

▶ Run

↻ Share

\$

Command

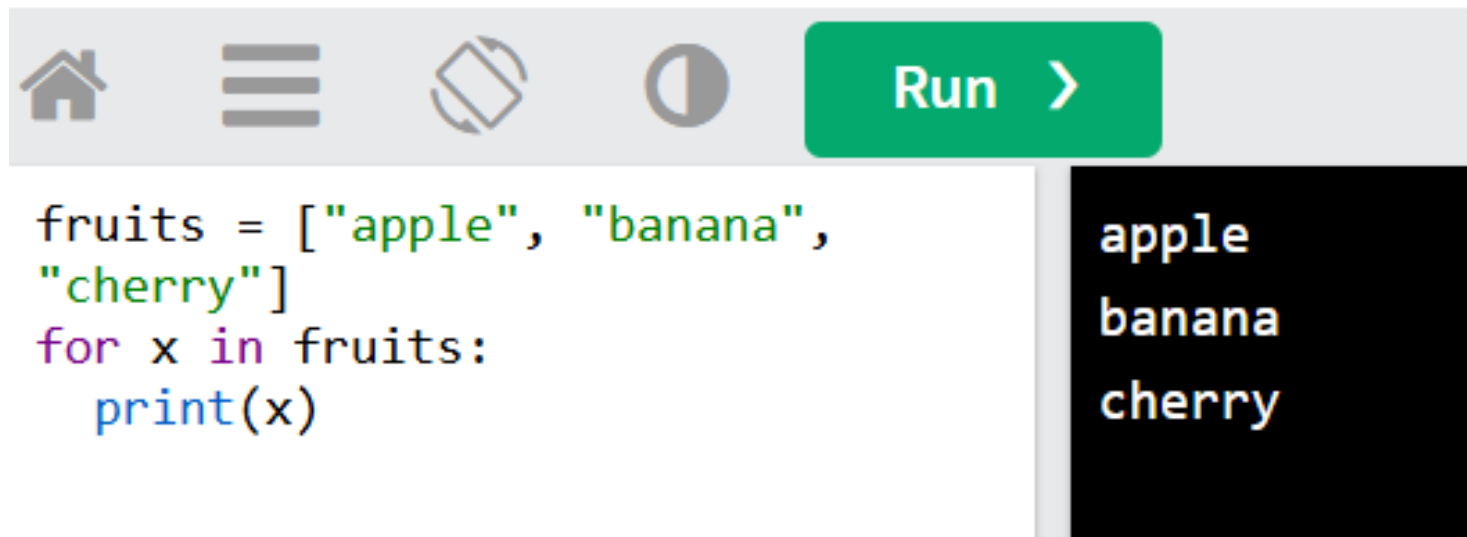
```
1
2 for x in range(50,350,100):
3     print("Number =", x)
4
```

📄

Number = 50  
Number = 150  
Number = 250

# Using the "In" Command with the For Loop

The **"in"** command in a for loop is used to check if a value exists within a sequence (like a list, string, or range).



The image shows a code editor interface with a toolbar at the top containing icons for home, menu, undo, and redo, followed by a green 'Run' button with a right arrow. Below the toolbar, the following Python code is displayed:

```
fruits = ["apple", "banana",  
"cherry"]  
for x in fruits:  
    print(x)
```

To the right of the code, a black output window displays the results of the loop:

```
apple  
banana  
cherry
```

# In Example- Lets' Try

---

ivebsim\_pythoncode >  for2.py > ...

```
1  country=["Türkiye","Poland","Romania","Portugal","Italy"]
2  for x in country:
3      print(x)
4
```

Türkiye

Poland

Romania

Portugal

Italy

PS C:\Users\semra\OneDrive\Masaüstü\ivebsim\_pythoncode> [

# While Loop

---

**We use the while loop** when we want a block of code to **repeat as long as a condition is true**.



If the number of repetitions is **not known in advance** and depends on a condition, the while loop is the best choice.

# While Loop

---

That is why it is commonly used in games, guessing programs, and applications that require continuous user input.



The while loop first checks the condition; if the condition is true, it executes. If the condition is false, the loop stops.

# While Loop

---

## Syntax:

```
while condition:
```

```
    # Code to be executed
```

```
    ....
```

```
continue from here
```

```
....
```

```
...
```

# Example: Counting from 1 to 5 using a while loop

---



## Python Code

```
python
```

```
counter = 1
```

```
while counter <= 5:
```

```
    print(counter)
```

```
    counter = counter + 1
```

## ▶ Step-by-Step Explanation

### ◆ Step 1: Create a variable

```
python  
  
counter = 1
```

We create a variable named **counter** and give it the value **1**.  
This variable will control the loop.

### ◆ Step 2: Check the condition

```
python  
  
while counter <= 5:
```

Python checks the condition:

➡ Is counter less than or equal to 5?

- If **YES**, the loop runs
- If **NO**, the loop stops

At the beginning:

✓ counter = 1 → condition is **True**

### ◆ Step 3: Execute the code inside the loop

```
python  
  
print(counter)
```

Python prints the current value of `counter`.

---

Output:

```
1
```

---

### ◆ Step 4: Update the variable

```
python  
  
counter = counter + 1
```

We increase `counter` by 1.

Now:

```
➡ counter = 2
```

This step is **very important**, otherwise the loop will never stop.

### ◆ Step 5: Repeat the loop

Python goes back to the `while` line and checks the condition again.

| counter value | condition ( $\leq 5$ ) | result     |
|---------------|------------------------|------------|
| 1             | True                   | loop runs  |
| 2             | True                   | loop runs  |
| 3             | True                   | loop runs  |
| 4             | True                   | loop runs  |
| 5             | True                   | loop runs  |
| 6             | False                  | loop stops |

### ● Step 6: Loop ends

When `counter` becomes 6, the condition is **False**.

The loop stops and the program ends.

# While Loop - Example

```
number = 1

while number <= 5:
    print(number)
    number += 1
```

```
1
2
3
4
5
```

```
count = 1

while count <= 3:
    print("Hello Python")
    count += 1
```

```
Hello Python
Hello Python
Hello Python
```

```
num = 0

while num <= 10:
    print(num)
    num += 2
```

```
0
2
4
6
8
10
```

# While Loop – Lets' Try

---

 while.py X

ivebsim\_pythoncode >  while.py > ...

```
1 password=""
2 while password!="python":
3     password=input("Enter password : ")
4 print("Access granted")
5
```

# For Loop vs While Loop

| Feature               | For Loop                                             | While Loop                                    |
|-----------------------|------------------------------------------------------|-----------------------------------------------|
| Purpose               | Repeats code a <b>known number of times</b>          | Repeats code <b>while a condition is true</b> |
| Number of repetitions | <b>Known in advance</b>                              | <b>Not known in advance</b>                   |
| Condition check       | Based on a sequence (range, list)                    | Based on a logical condition                  |
| Loop ending           | Ends <b>automatically</b> when the sequence finishes | Ends when the condition becomes <b>false</b>  |
| Common usage          | Counting, lists, ranges                              | Games, password checks, user input            |
| Risk of infinite loop | Low                                                  | Higher if condition is not updated            |
| Example usage         | <pre>for i in range(5):</pre>                        | <pre>while x &lt; 5:</pre>                    |
| Best choice when...   | You know how many times to repeat                    | You don't know when to stop                   |



---

Congratulations