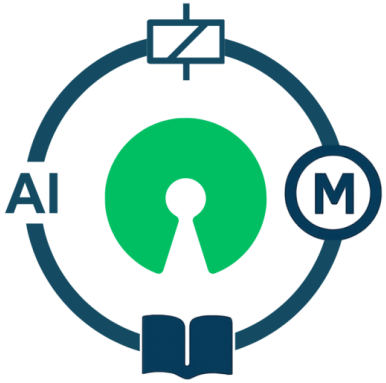


Open Source Simulator Software

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.

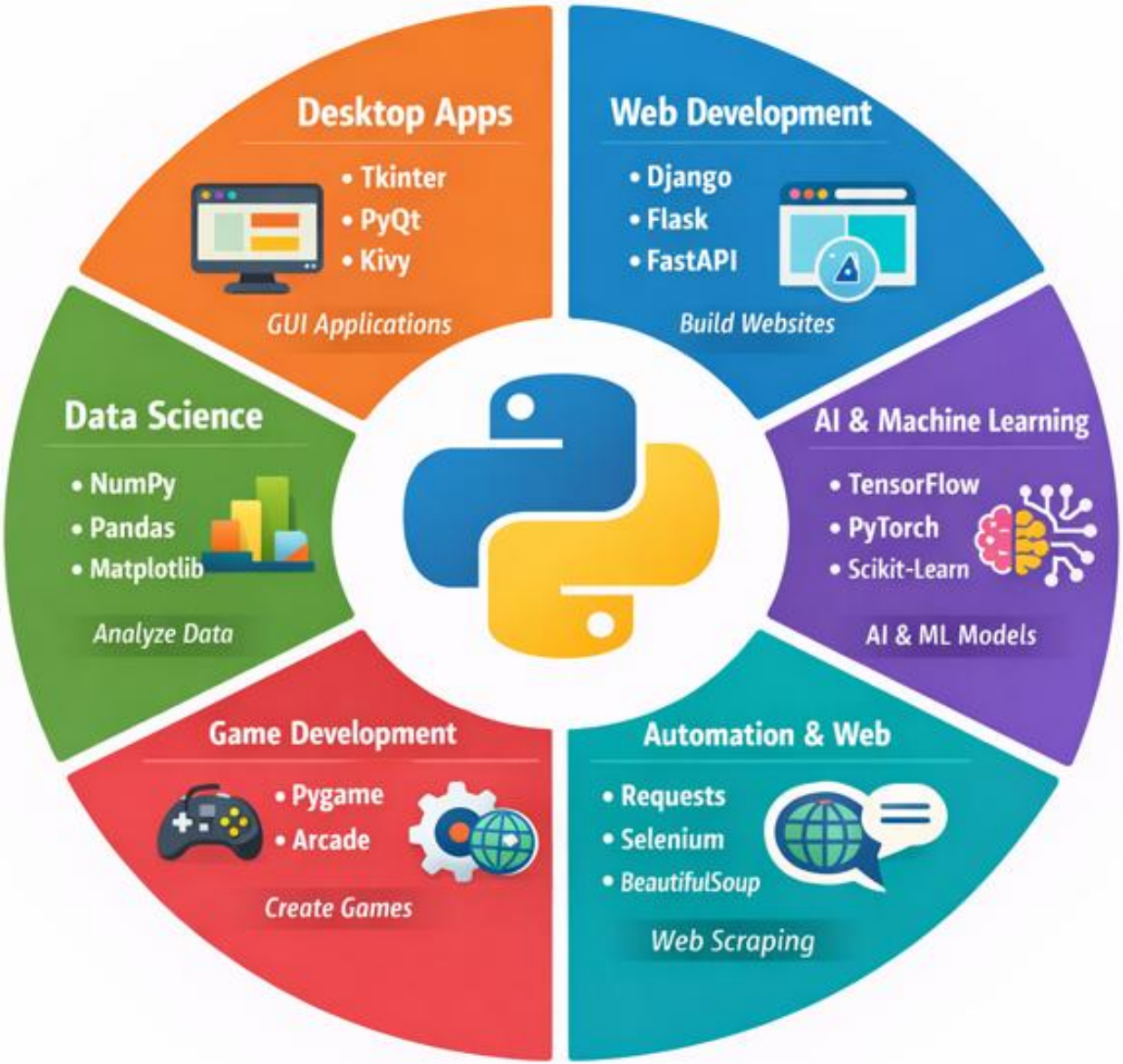
It uses indentation (space at the beginning of the line) to define the scope of loops, functions, and classes.

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

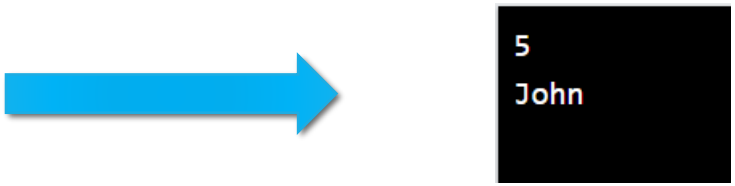
Variables store data in programs

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

Codes:

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

Output



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 alpha-numeric 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=’Yılmaz’

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]

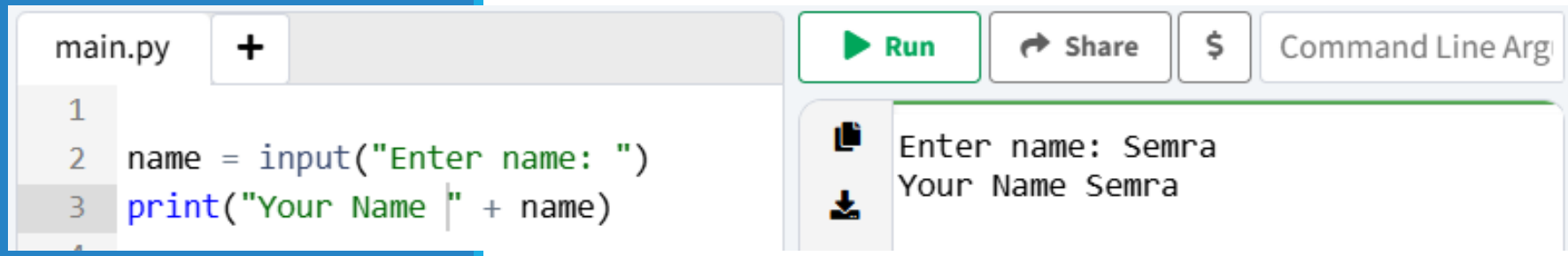
Input and Output

INPUT() GETS DATA

PRINT() SHOWS OUTPUT

Input Variables

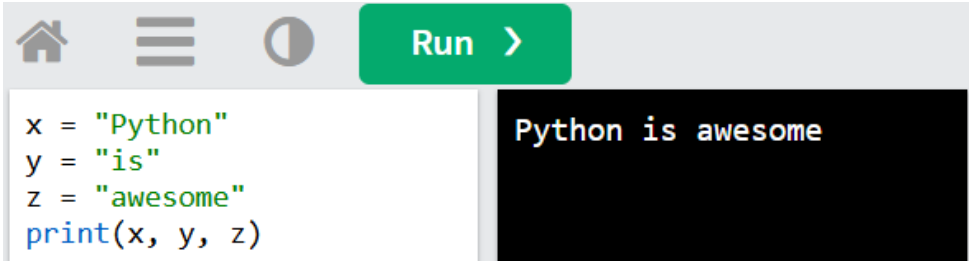
- We use the input() command to get a value from the user.
- We assign the received value to a variable.



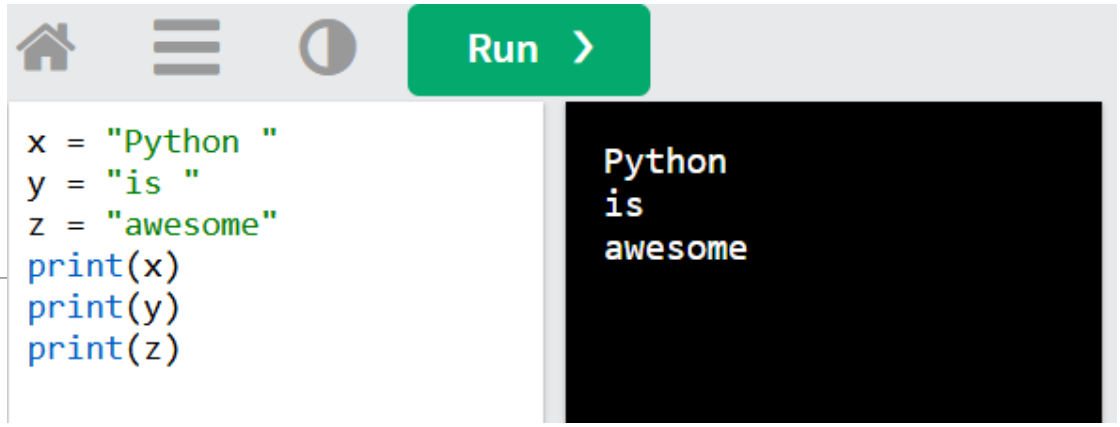
Output Variables

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

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



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



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

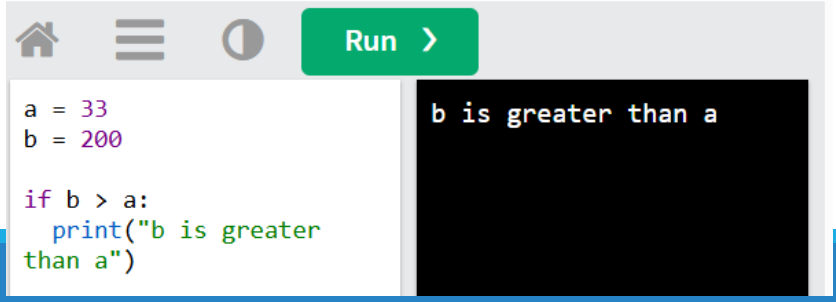
Example

If statement:

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

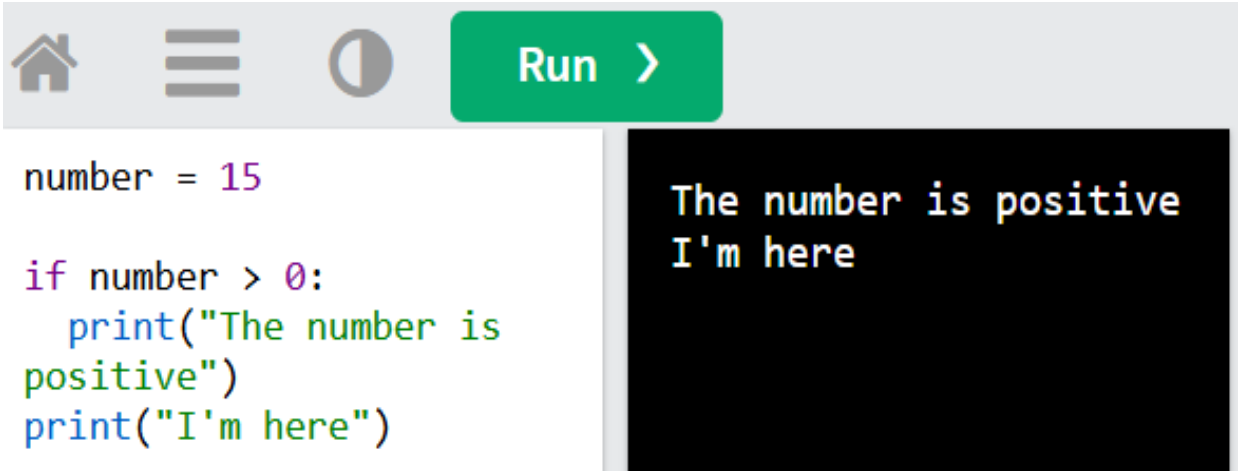
In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a.

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**".

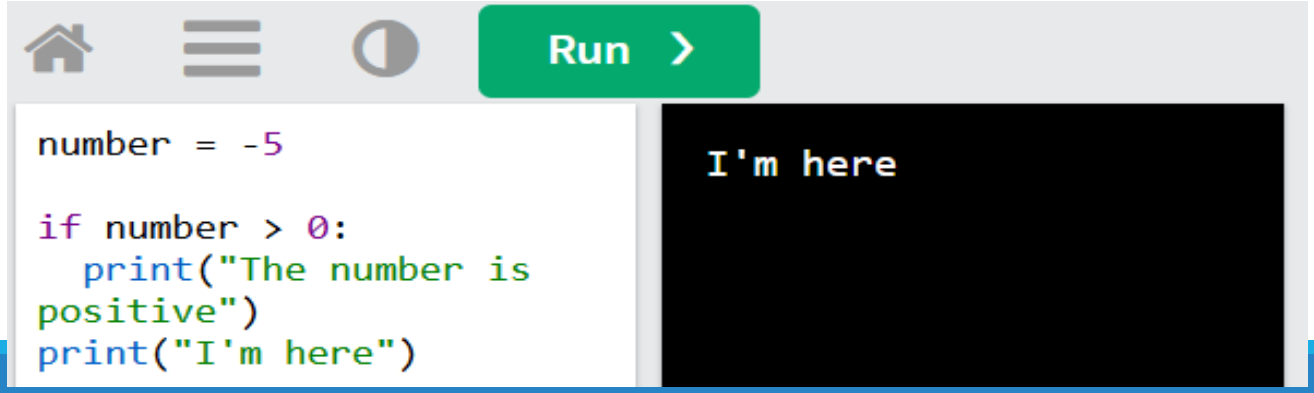


If Example

- For number= 15;



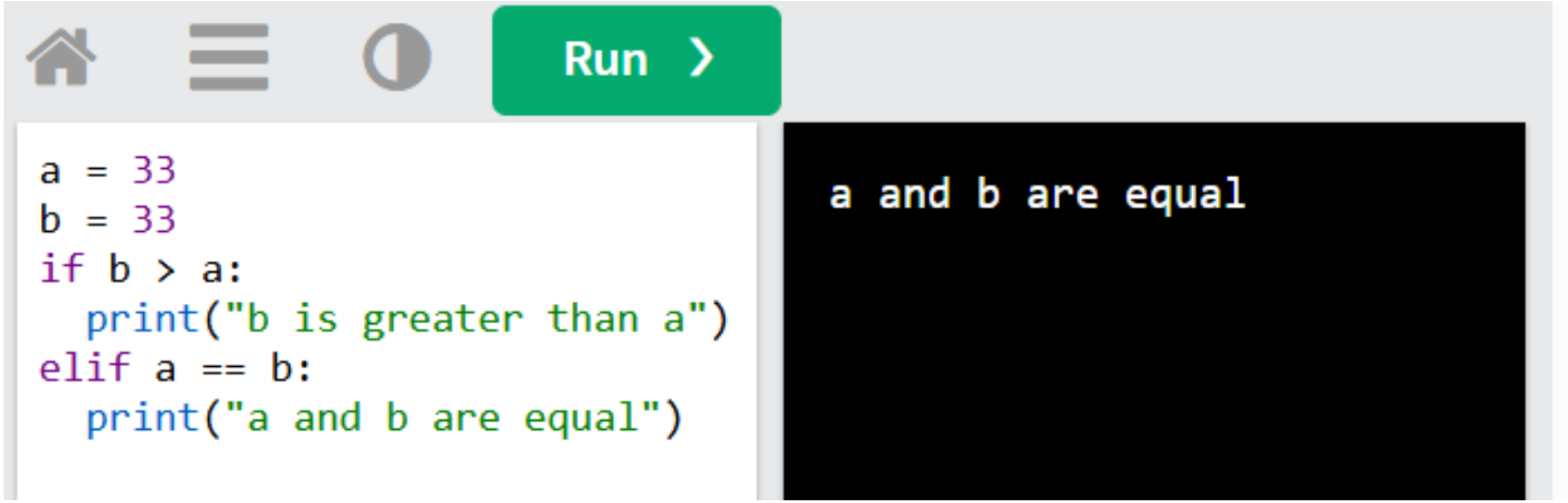
- For number= -5;



Python Elif Statement

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

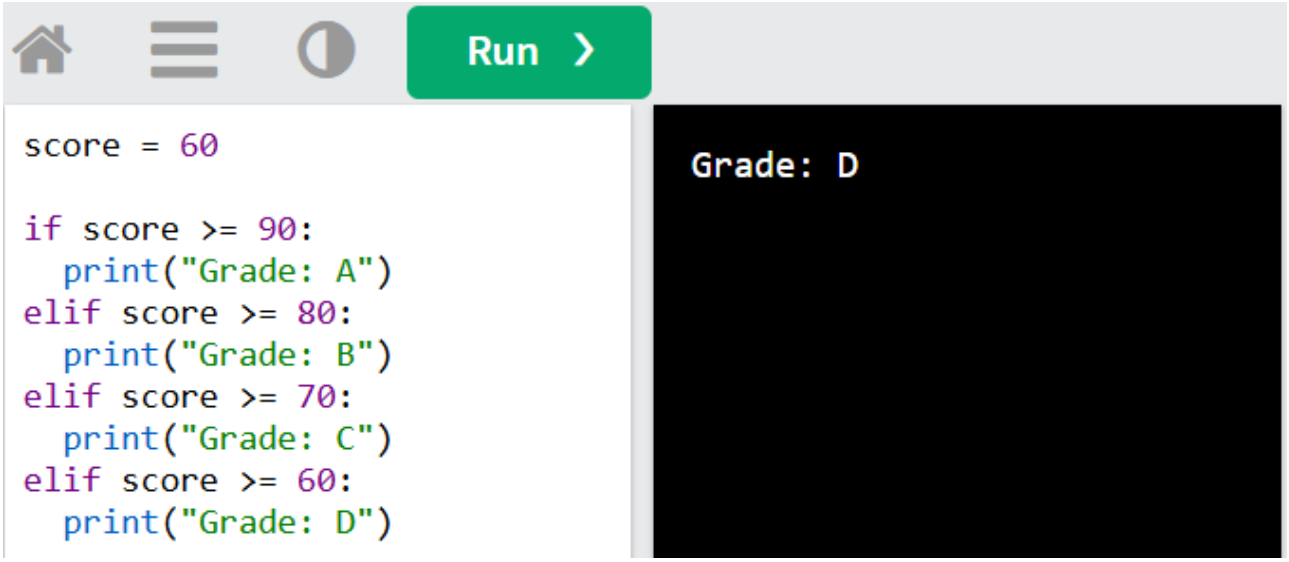
Example



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.



İf 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.

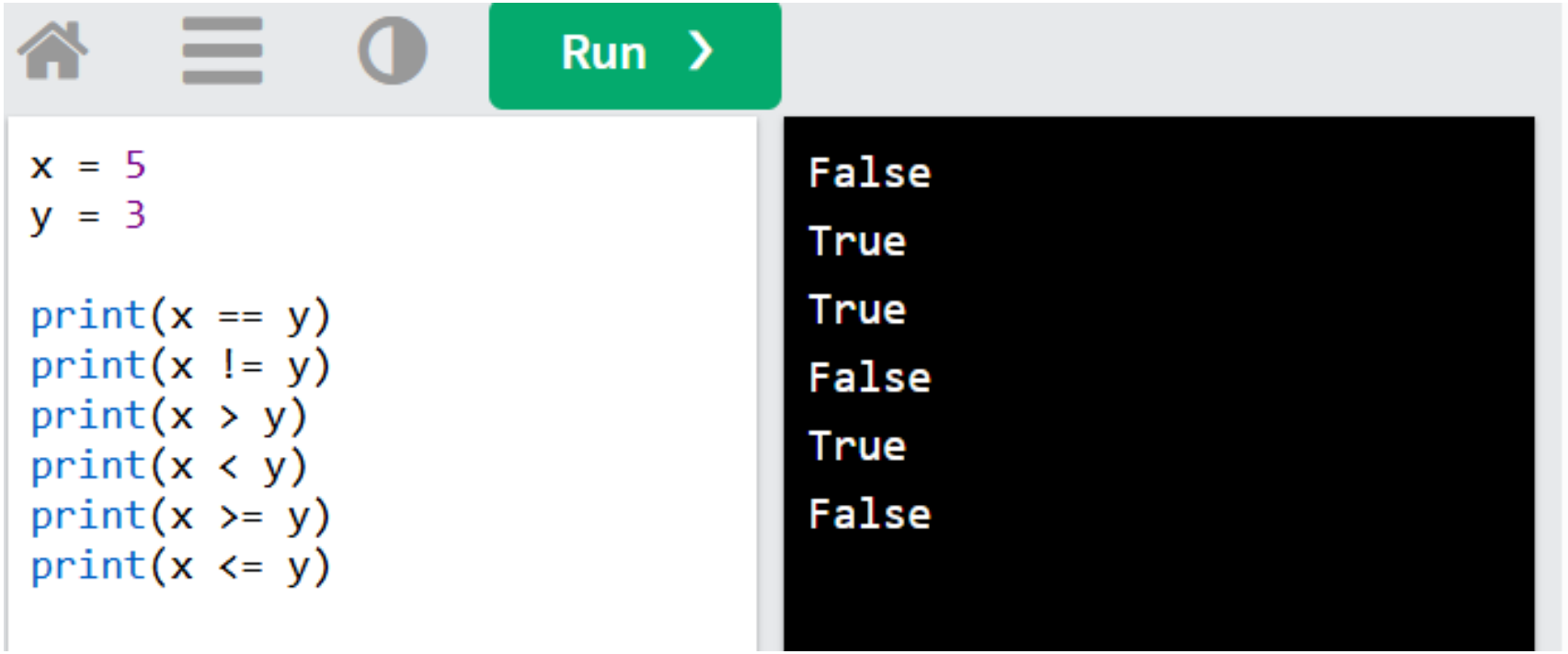
Operators are used in decision making and used in Loops

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



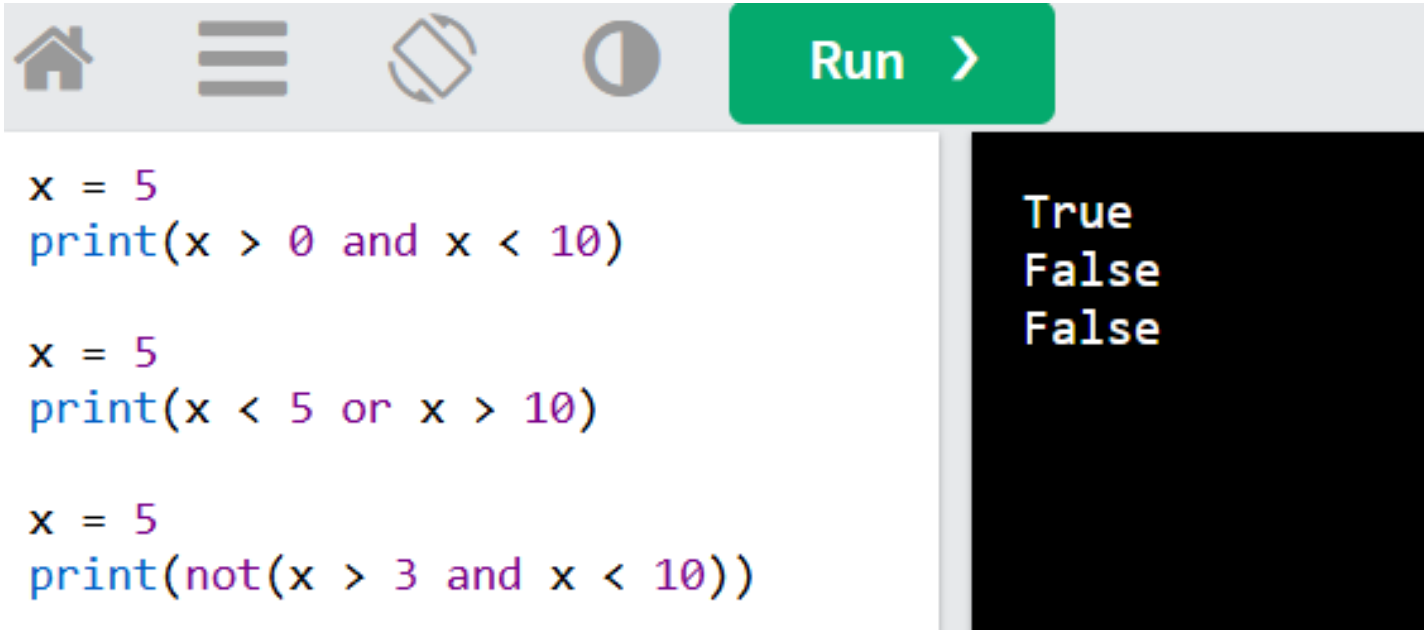
Logical operators

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	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Logical operators - Examples

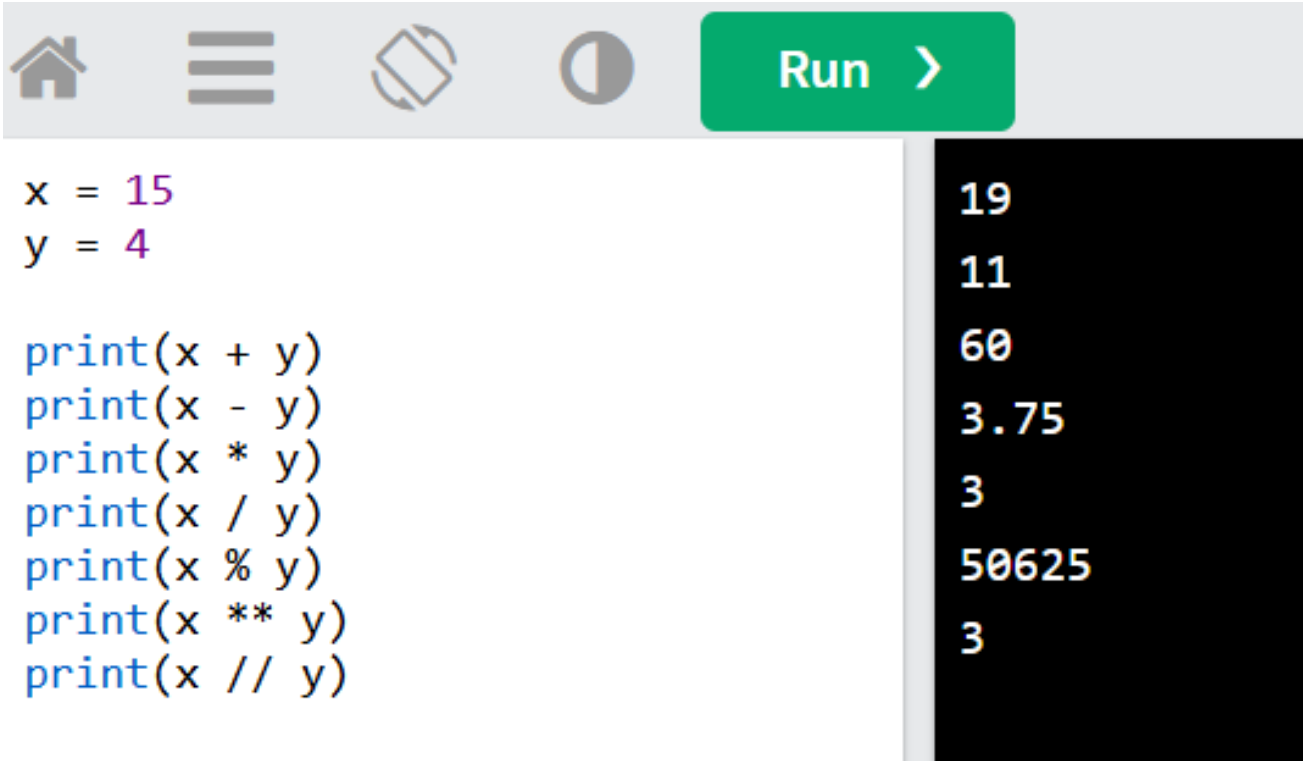


Arithmetic Operators

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



LOOPS

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.

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)

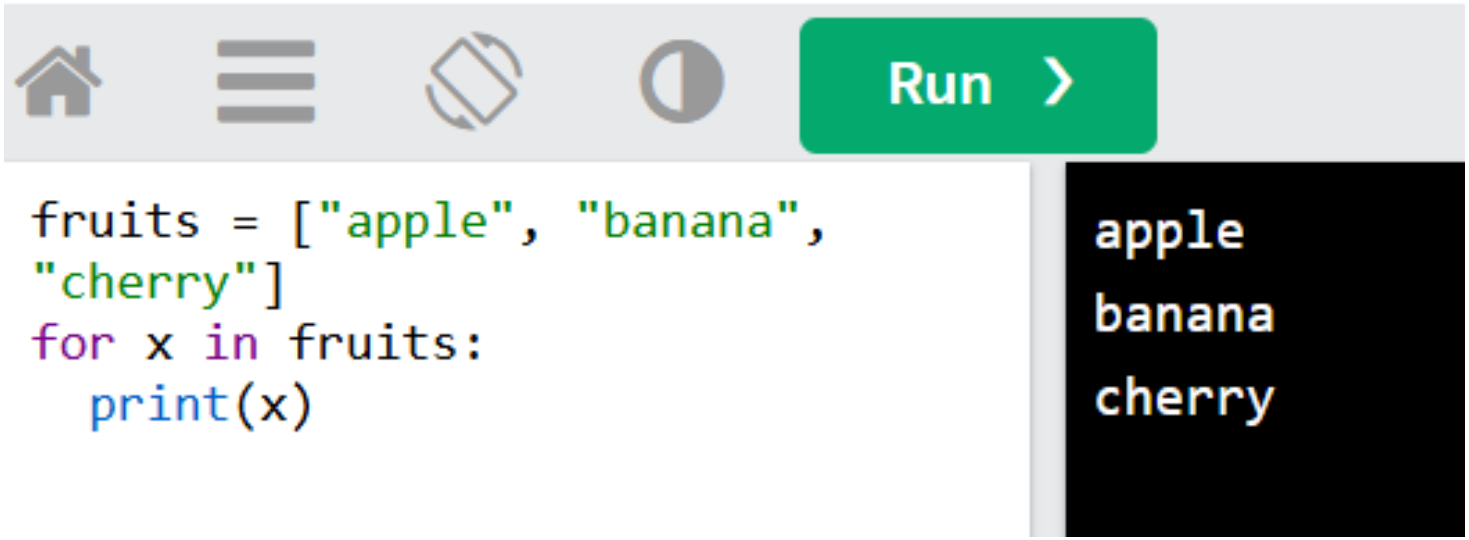
Range Examples

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

Range Example- Lets' Try

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).



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 (<=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

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

Congratulations