Imagine you're giving instructions to a robot to complete a task.

You need to tell it step by step what to do in a precise way because the robot doesn't understand ambiguity.

This is exactly what programming is—telling a computer what to do in a language it understands.

**Programming** is the process of writing instructions (called code) for a computer to perform tasks, solve problems, or automate processes.

Think of code as a set of recipes. The ingredients are variables and data, and the instructions (steps) are operations and logic.

Humans[High Level Language] ---> Compiler/Interpreter ----> Machines[0,1 Low/Machine Level Lang]

Python is a high-level, **interpreted** programming language meaning (you don't need to compile it before running.)

**Compiler and Interpreter**

Compiler -> Converts the entire source code into machine code before execution

Interpreter -> Translates and executes the source code line by line.)

When to use Compiler[Compiled Language]---> C Vs Interpreter[Interpreted Lang]--> Python?

It was created in the late 1980s by **Guido van Rossum** and was released in 1991.

It's designed to be **readable**, **concise**, and **easy to use**, which makes it accessible for beginners and powerful for experienced developers.

The language is named after the BBC show “Monty Python’s Flying Circus” and has nothing to do with reptiles. Making references to Monty Python skits in documentation is not only allowed, it is encouraged!

```
from IPython.display import YouTubeVideo

YouTubeVideo('Fh3ghPC-oEQ', width=800, height=300)
```


Python is a high-level language, which means it abstracts away the complexities of the computer hardware.

This makes it easy for humans to write but requires more processing by the computer to run.

- It is also an interpreted language, which means that Python code is executed line by line by the
- Python interpreter, unlike languages like C or Java, which are compiled into machine code before they are run.
- Python supports multiple programming paradigms:
  - Procedural Programming: Organizing code into reusable procedures (functions).
  - Object-Oriented Programming (OOP): Creating objects that encapsulate both data and functions.
  - Functional Programming: Treating computation as the evaluation of mathematical functions.

Python's popularity comes from several key factors:

- **Simple Syntax**: Python's syntax is clean and easy to understand, which makes it an ideal language for beginners.
- **Versatility**: Python is used in many different fields—web development, data science, machine learning, automation, game development, and more.
- **Huge Community**: Python has one of the largest communities, so it's easy to find resources, libraries, and frameworks to help with projects.
- **Extensive Libraries**: Python has libraries for nearly everything, from data manipulation (pandas, numpy) to machine learning (scikit-learn, TensorFlow) to web development (Django, Flask).
- **Cross-Platform**: Python runs on all major platforms (Windows, macOS, Linux), so you can write a Python program on one system and run it on another without changing the code.
- **Jobs**: With its widespread use in web development, automation, and especially in data science and artificial intelligence, Python skills are highly in demand.

# Python Programming Basics - 1

We will dive into the basic syntax and constructs of Python. We will cover:

1. Variables
2. Data Types
3. Operators

### Python Syntax

Python is known for its simple and readable syntax. Let's look at a simple Python script.

```
# Print a simple message
print("Hello, Python!")
```

```
Hello, Python!
```

## Variables and Data Types

A variable in Python is a reserved memory location to store values. Python automatically determines the type of the variable based on the assigned value.

```
# assign a variable
a = 10
b = 23

# print the values using an f string (formatted string literals)
```

```
print(f" a is {a} and b is {b}")
print("a is {} and b is {} and their sum is {}".format(a, b, a + b)) # uses the .format() method
print("a is %d and b is %d and their sum is %d" % (a, b, a + b)) # % Formatting (Old Style)


 a is 10 and b is 23
a is 10 and b is 23 and their sum is 33
a is 10 and b is 23 and their sum is 33
```

## Data Types:

- Python has several built-in data types that are fundamental to its functionality. Here's a quick overview of the most commonly used data types:

1. Numbers: These include integers and floating-point numbers.
   - Integers `int`: Whole numbers (e.g., 5, -3, 42)
   - Floats `float`: Numbers with a decimal point (e.g., 3.14, -0.001)
2. Strings `str`: Sequences of characters enclosed in single quotes, double quotes, or triple quotes.
   - Single-line String: "Hello"
   - Multi-line String: """Hello, World!"""

# 3. Booleans `bool`: Represents True or False values.

4. Lists: Ordered, mutable collections of items, which can be of mixed types.
5. Tuples: Ordered, immutable collections of items.
6. Dictionaries: Unordered collections of key-value pairs.
7. Sets: Unordered collections of unique items.
8. None: Represents the absence of a value or a null value.

- Each data type serves different purposes and has different methods associated with it. For example, lists and dictionaries have methods to modify their content, while tuples and strings do not.

## Floating-Point Numbers (float)

(Only covering this topic, because important from interview pov)

Definition: Floating-point numbers are numbers with a decimal point. They can represent fractions and very large or very small numbers.

```
pi = 3.14159          # Decimal number
negative = -0.1       # Negative decimal number
scientific = 1.23e4   # 12300.0 in scientific notation
```

**Characteristics:**

- Precision: Floating-point numbers are subject to precision limitations.
- Python uses double-precision (64-bit) floating-point numbers, which may lead to small rounding errors.

1. Arithmetic Operations:
   - Like integers, you can perform arithmetic operations on floating-point numbers.
2. Division:
   - Division of integers where one or both operands are floats results in a float.
3. Formatting:
   - You can format floating-point numbers to control the number of decimal places displayed.
4. Methods:
   - is_integer(): Checks if the float has an integer value.
   - as_integer_ratio(): Returns a tuple representing the float as a fraction.

```
#Typecasting
number = 5.0
# type(number)
print(number.is_integer())  # True
number
```

```
True
```

```
5.0
```

```
ratio = (0.75).as_integer_ratio()  # (3, 4)
ratio
```

```
(3, 4)
```

## Declaring Variables

```
# Integer
x = 10
print("x is of type:", type(x))

# Float
y = 3.14
print("y is of type:", type(y))

# String
name = "John"
print("name is of type:", type(name))
```

```
# Boolean
is_active = True
print("is_active is of type:", type(is_active))
```

```
x is of type: <class 'int'>
y is of type: <class 'float'>
name is of type: <class 'str'>
is_active is of type: <class 'bool'>
```

```
# Example
a = 10        # Positive integer
b = -3        # Negative integer
c = 0         # Zero

sum = 5 + 3         # 8
difference = 10 - 2 # 8
product = 4 * 7     # 28
quotient = 20 / 4   # 5.0 (Note: Division always returns a float)

# setting precision of float numbers
formatted = "{:.7f}".format(3.14159)  # 3.14
formatted
```

## Type Conversion

**Type casting**, also known as type conversion, is a process in programming where you convert a variable from one data type to another.

You can convert between data types using type conversion functions like `int()`, `float()`, `str()`, etc.

```
# Convert float to int
num = 5.67
num_int = int(num)
print("Converted to int:", num_int)

# Convert int to float
age = 25
age_float = float(age)
print("Converted to float:", age_float)

# Convert int to string
score = 90
score_str = str(score)
print("Converted to string:", score_str)
```

```
Converted to int: 5
Converted to float: 25.0
Converted to string: 90
```

# Operators in Python

Operators are used to perform operations on variables and values.

**Arithmetic Operators**

Common arithmetic operators include: `+`, `-`, `*`, `/`, `//` (floor division), `%` (modulus), and `**` (exponentiation).

1. Division (/):
   - Always results in a floating-point number.

1. Floor Division (//):
   - Divides and returns the largest integer less than or equal to the result.

1. Modulus (%):
   - Returns the remainder of the division.

1. Exponentiation (**):
   - Raises a number to the power of another.

1. Methods:
   - bit_length(): Returns the number of bits necessary to represent the integer in binary.

```
# Arithmetic operations
a = 10
b = 3

print("Addition:", a + b)
print("Subtraction:", a - b)
print("Multiplication:", a * b)
print("Division:", a / b)

print("Floor Division:", a // b)
print("Modulus:", a % b)
print("Exponentiation:", a ** b)
```

```
Addition: 13
Subtraction: 7
Multiplication: 30
Division: 3.3333333333333335
Floor Division: 3
```

```
Modulus: 1
Exponentiation: 1000

7

bits = (100).bit_length()  # 4 --> 1000 binary 0,1
bits
```

**Comparison Operators**

Comparison operators include: `==`, `!=`, `>`, `<`, `>=`, `<=`. These return a boolean value (`True` or `False`).

```
# Comparison operations
x = 10
y = 20

print("Is x equal to y?", x == y)
print("Is x not equal to y?", x != y)
print("Is x greater than y?", x > y)
print("Is x less than or equal to y?", x <= y)

Is x equal to y? False
Is x not equal to y? True
Is x greater than y? False
Is x less than or equal to y? True

False
```

**Logical Operators**

Logical operators are used to combine conditional statements.

- `and`: Returns True if both statements are true
- `or`: Returns True if one of the statements is true
- `not`: Reverses the result

```
# Logical operations
x = 5
y = 10

print("x > 0 and y > 0:", x > 0 and y > 0)
print("x > 0 or y < 0:", x > 0 or y < 0)
print("not(x > 0):", not(x > 0))

x > 0 and y > 0: True
x > 0 or y < 0: True
not(x > 0): False
```

Now it's your turn!

## Task 1: Variable Assignment

1. Create a variable `a` and assign it the value `50`.
2. Create another variable `b` and assign it the value `30.5`.
3. Print their values and types.

## Task 2: Operators

1. Using variables `a` and `b` from Task 1, perform the following operations:
   - Addition
   - Subtraction
   - Multiplication
   - Floor Division
2. Compare the two variables to check if `a` is greater than `b`.
3. Swap the values of `a` and `b` without using a third variable.