INTRODUCTION TO PYTHON

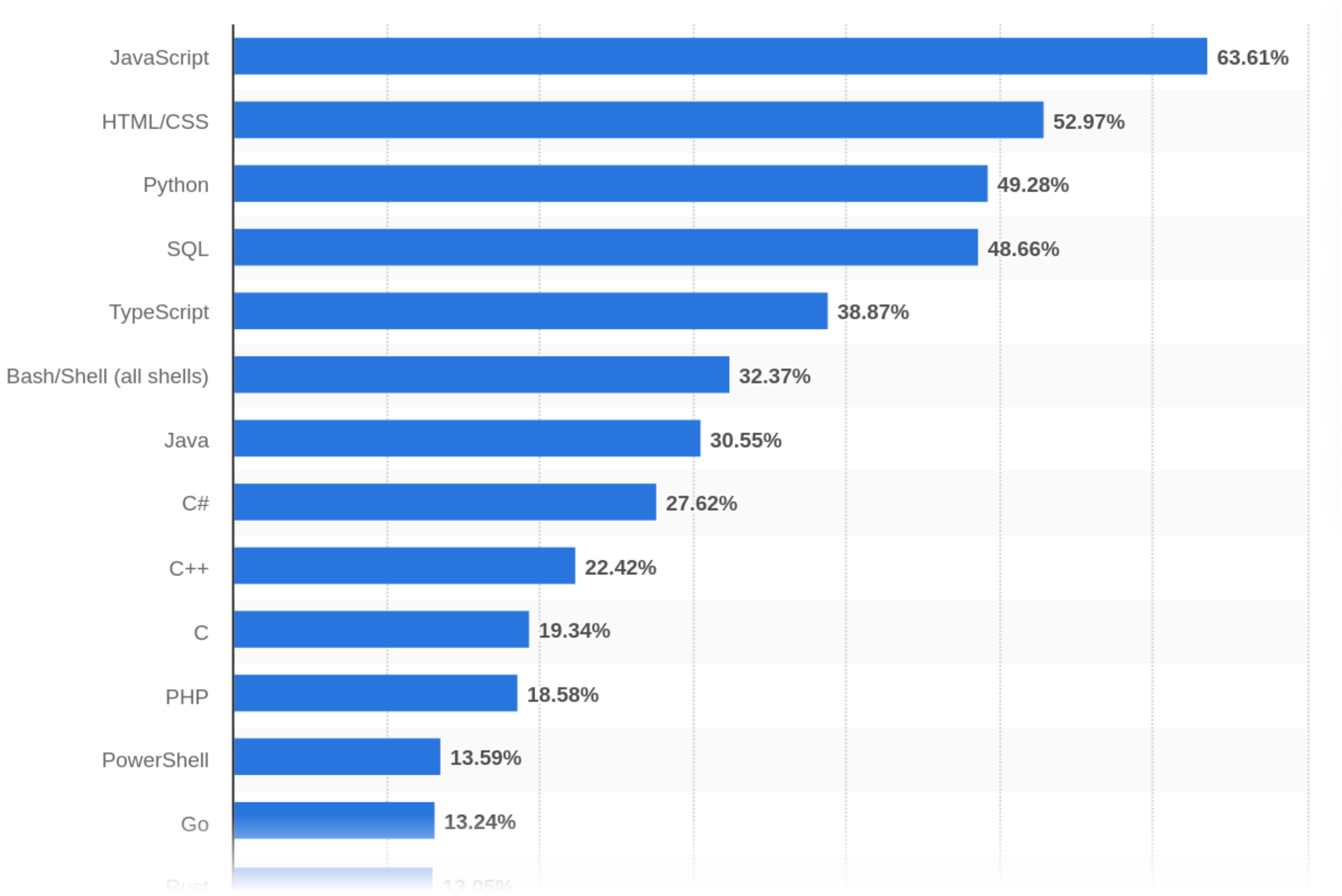
WHAT IS PYTHON?

- Python is a high-level, interpreted programming language.
- Known for its clear syntax and readability.
- Developed by Guido van Rossum in the late 1980s.

POPULARITY AND USAGE

- One of the most popular programming languages in the world.
- Used in web development, data science, artificial intelligence, and more.
- Loved by beginners and professionals alike.

MOST USED PROGRAMMING LANGUAGES 2023



INTERPRETED VS. COMPILED LANGUAGES

- Python is an interpreted language, meaning code is executed line by line.
- Compiled languages (like C#) convert code into machine language before execution, leading to faster runtime.
- Each approach has its advantages: Python for ease of use and rapid development, compiled languages for performance.

WHY LEARN PYTHON?

- Easy to learn for beginners.
- Extensive libraries and frameworks.
- Strong community support and vast resources.
- Great for data science and machine learning.

SETTING UP THE ENVIRONMENT

INSTALLING PYTHON

- Download Python from the official website.
- Choose the version suitable for your operating system.
- Follow the installation guide, ensuring that Python is added to your system's PATH.

INTRODUCTION TO VS CODE

- VS Code is a free, open-source editor by Microsoft.
- Highly customizable and supports numerous programming languages.
- Extensive marketplace for extensions, including Python support.

SETTING UP PYTHON IN VS CODE

- Install VS Code from the official site.
- Open VS Code and navigate to the Extensions view (`Ctrl+Shift+X`).
- Search for and install the ‘Python’ extension by Microsoft.

CREATING YOUR FIRST PYTHON PROJECT

- Open VS Code and create a new folder for your project.
- Create a new Python file (`.py` extension).
- Write a simple Python script, like printing a message.

RUNNING PYTHON CODE IN VS CODE

- Open your Python file in VS Code.
- You can run the script by right-clicking in the editor and selecting 'Run Python File in Terminal'.
- Observe the output in the integrated terminal.

HELLO WORLD (YOUR FIRST PYTHON PROGRAM)

HELLO WORLD

- The first program you write in any programming language is usually a simple message.
- In Python, this is done using the `print()` function.
- Create a new Python file using VS Code and write the following code:

```python
print("Hello World!")
```

HELLO WORLD

- Save the file as `hello_world.py`.
- Run the script by right-clicking in the editor and selecting 'Run Python File in Terminal'.
- Observe the output in the integrated terminal.

HELLO WORLD

- You can also run the script by opening the integrated terminal and typing

```
python hello_world.py
```

- This is the preferred method for running Python scripts.

COMMENTS

- Comments are used to explain code and make it more readable.
- In Python, comments are created using the # symbol.
- Comments are ignored by the interpreter.

```python
# This is a comment
print("Hello World!") # This is also a comment
```

USER INPUT

INTRODUCTION TO USER INPUT

- User input is used to get data from the user.
- In Python, you can use the `input()` function to prompt the user for input.
- The `input()` function returns a string containing the user's input.
- The parameter passed to the `input()` function is the prompt message.

```python
name = input("Enter your name: ")
print("Hello " + name)
```

VARIABLES AND DATA TYPES

INTRODUCTION TO VARIABLES

- Variables are used to store data in a program.
- In Python, you can create a variable by assigning a value to it.
- The value can be of different data types, such as numbers, strings, or booleans.
- In Python you don't need to specify the data type of a variable like in other languages.

VARIABLE ASSIGNMENT

- You can assign a value to a variable using the assignment operator (=).
- The value can be a literal or the result of an expression.
- Variables can be reassigned with a new value at any time.

```
# Assigning a number to a variable
x = 5
# now 5 is stored in x
```

NUMERIC DATA TYPES

- Python supports various numeric data types, including integers (`int`), floating-point numbers (`float`), and complex numbers (`complex`).
- You can perform arithmetic operations on numeric data types, such as addition, subtraction, multiplication, and division.

```python
a = 5
b = 2.5
result = a + b
print(result) # 7.5
```

STRING DATA TYPE

- Strings are sequences of characters enclosed in single or double quotes.
- You can perform various operations on strings, such as concatenation, slicing, and formatting.
- Python provides a rich set of string manipulation methods.

```python
firstname = "Alexander"
lastname = "Kornfellner"
fullname = firstname + " " + lastname # concatenation
```

BOOLEAN DATA TYPE

- Boolean data type represents truth values, either `True` or `False`.
- Boolean values are often used in conditional statements and logical operations.
- Python provides logical operators, such as `and`, `or`, and `not`, for working with boolean values.

```
is_valid = True
is_admin = False
```

TYPE CONVERSION

- Python allows you to convert data from one type to another.
- This can be done using built-in functions like `int()`, `float()`, `str()`, and `bool()`.
- Type conversion is useful when you need to perform operations on different data types.

```python
x = "5" # this is a string!
y = int(x)  # convert to integer

name = "Alex"
z = int(name) # this will cause an error
```

VARIABLE NAMING RULES

- Variable names in Python can contain letters, numbers, and underscores.
- They must start with a letter or an underscore.
- Variable names are case-sensitive.
- Avoid using reserved keywords as variable names.
- in good practice variable names should be descriptive and writte in `snake_case`

VARIABLE SCOPE

- The scope of a variable determines where it can be accessed in a program.
- In Python, variables have either global or local scope.
- Global variables can be accessed from anywhere in the program.
- Local variables are defined within a specific block or function.

OPERATORS AND EXPRESSIONS

- Operators are symbols that perform operations on operands.
- Python supports various types of operators, such as arithmetic, assignment, comparison, logical, and bitwise operators.
- Expressions are combinations of operators, operands, and variables that evaluate to a value.

ARITHMETIC OPERATORS

- Arithmetic operators are used to perform mathematical operations on numeric data types.
- Python supports the following arithmetic operators:
  - `+` Addition
  - `-` Subtraction
  - `*` Multiplication
  - `/` Division
  - `%` Modulus
  - `**` Exponentiation
  - `//` Floor division

ASSIGNMENT OPERATORS

- Assignment operators are used to assign values to variables.
- Python supports the following assignment operators:
  - $=$ Assign
  - $+=$ Add and assign
  - $-=$ Subtract and assign
  - $*=$ Multiply and assign
  - $/=$ Divide and assign

COMPARISON OPERATORS

- Comparison operators are used to compare values.
- Python supports the following comparison operators:
  - $==$ Equal to
  - $!=$ Not equal to
  - $>$ Greater than
  - $<$ Less than
  - $>=$ Greater than or equal to
  - $<=$ Less than or equal to

LOGICAL OPERATORS

- Logical operators are used to combine boolean values and perform logical operations.
- Python supports the following logical operators:
    - `and` Logical AND (both operands must be true)
    - `or` Logical OR (at least one operand must be true)
    - `not` Logical NOT

PRECEDENCE AND ASSOCIATIVITY

- Operators in Python have different precedence levels, which determine the order of evaluation.
- Parentheses can be used to override the default precedence.
- Operators with the same precedence are evaluated from left to right, unless otherwise specified.

EXPRESSIONS

- Expressions are combinations of operators, operands, and variables that evaluate to a value.
- Python supports a wide range of expressions, including arithmetic expressions, logical expressions, and conditional expressions.

CODE BLOCKS

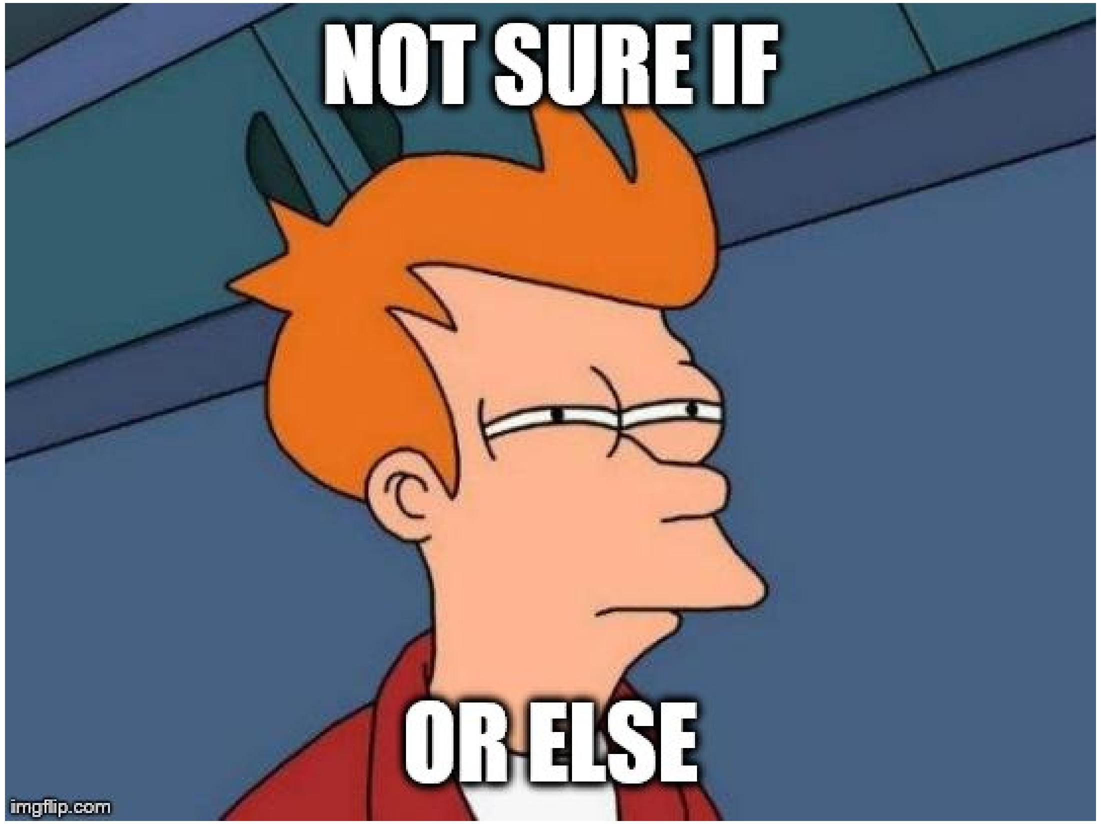
INTRODUCTION TO CODE BLOCKS

- Code blocks are used to group statements together.
- In Python, code blocks are defined using indentation.
- All statements within a code block must be indented by the same amount.

CODE BLOCKS (EXAMPLE)

```
if x > 5:
    print("x is greater than 5")
    print("x is a positive number")
```

CONDITIONS



UNDERSTANDING CONDITIONAL STATEMENTS

- Conditional statements allow you to execute different code based on certain conditions.
- The `if` statement is the simplest form of conditional execution.
- It evaluates a condition and executes the code block if the condition is true.

```python
x = 10
if x > 5:
    print("x is greater than 5")
```

THE `if-else` STATEMENT

- An `else` statement can follow an `if` statement.
- This is used to handle the case where the `if` condition is false.

```
y = 3
if y > 5:
    print("y is greater than 5")
else:
    print("y is not greater than 5")
```

THE `elif` STATEMENT

- `elif` (short for else if) is used for multiple conditions.

```python
z = 7
if z > 10:
    print("z is greater than 10")
elif z > 5:
    print("z is greater than 5 but not greater than 10")
else:
    print("z is 5 or less")
```

NESTED CONDITIONAL STATEMENTS

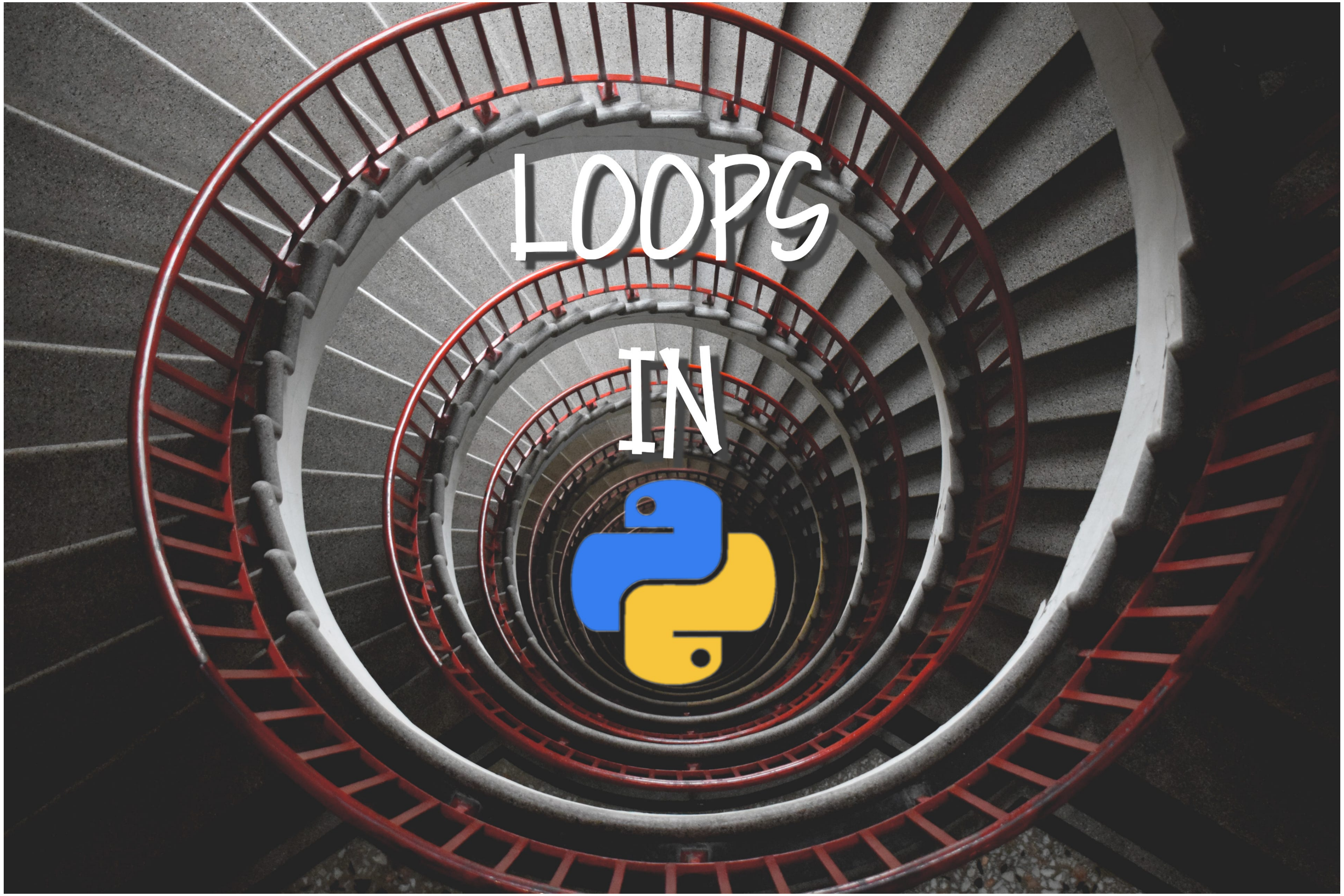
- You can nest `if` statements within each other for more complex conditions.

```python
a = 10
b = 20
if a > 5:
    if b > 15:
        print("a > 5 and b > 15")
    else:
        print("a > 5 but b <= 15")
```

CONDITIONAL EXPRESSIONS (TERNARY OPERATORS)

- Python supports conditional expressions, which are a compact way to use ` if-else` ` ` statements.

```python
age = 18
status = "adult" if age >= 18 else "minor"
print(status)
```


LOOPS
IN

INTRODUCTION TO LOOPS

- Loops are used to execute a block of code repeatedly.
- Python supports two types of loops: `for` loops and `while` loops.
- `for` loops are used to iterate over a sequence of values.
- `while` loops are used to execute a block of code while a condition is true.

THE `while` LOOP

- The `while` loop executes a block of code while a condition is true.
- The condition is evaluated before each iteration.
- The loop terminates when the condition becomes false.

```python
i = 0
while i < 5:
    print(i)
    i += 1
```

`range()` FUNCTION

- The `range()` function is used to generate a sequence of numbers.
- It takes three parameters: `start`, `stop`, and `step`.
- The `start` parameter specifies the starting value of the sequence (default is 0).
- The `stop` parameter specifies the ending value of the sequence (not included).
- The `step` parameter specifies the increment (default is 1).

range() FUNCTION

```
range(5) # [0, 1, 2, 3, 4]
range(1, 5) # [1, 2, 3, 4]
range(1, 10, 2) # [1, 3, 5, 7, 9]
```

THE `for` LOOP

- The `for` loop is used to iterate over a sequence of values.
- The loop variable is assigned a value from the sequence in each iteration.
- The loop terminates when all values in the sequence have been processed.

```python
for i in range(5): # range(5) returns [0, 1, 2, 3, 4]
    print(i)
```

`break`ING OUT OF A LOOP

- The `break` statement is used to exit a loop prematurely.
- It is often used to terminate a loop when a certain condition is met.

```python
for i in range(10):
    if i == 5:
        break
    print(i)
```

`continue`ING A LOOP

- The `continue` statement is used to skip the current iteration of a loop.
- It is often used to skip certain values in a sequence.

```python
for i in range(10):
    if i % 2 == 0:
        continue
    print(i)
```