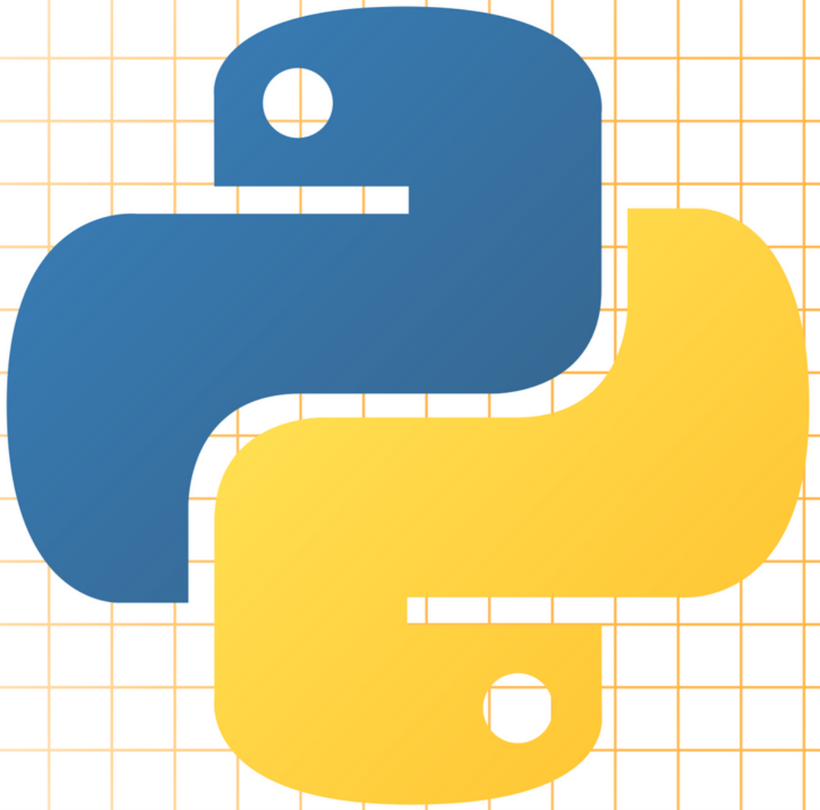
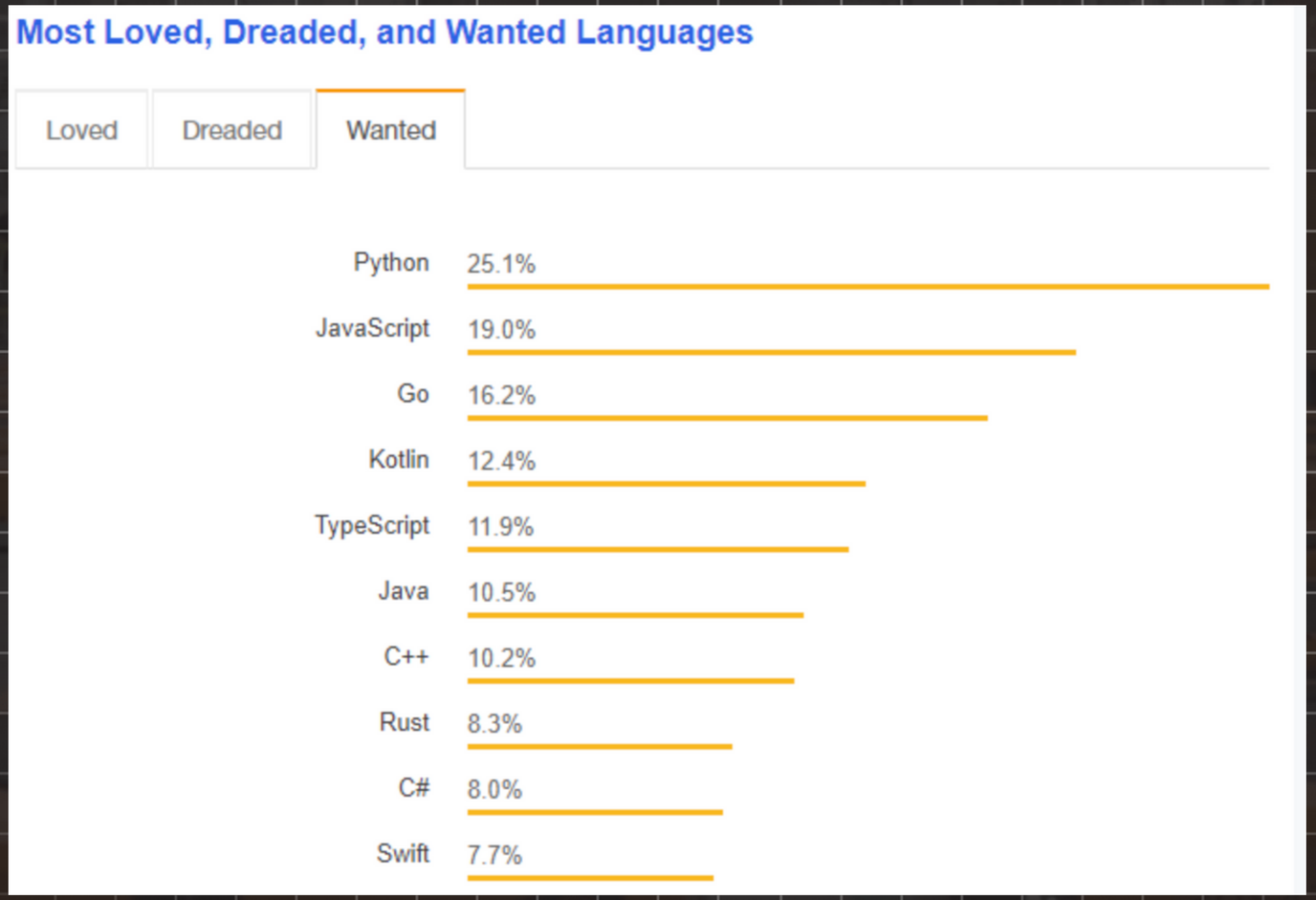
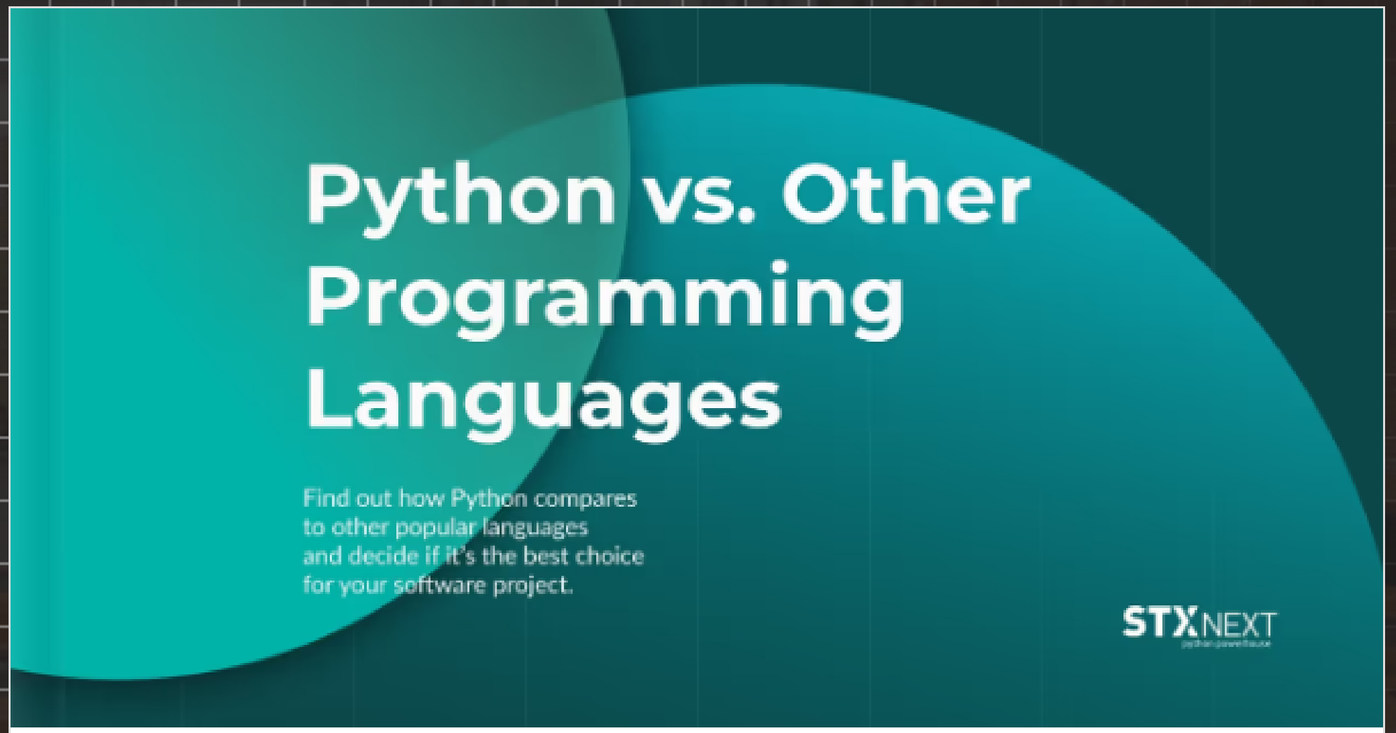


# Introduction to Python

# WHY PYTHON


Most Loved, Dreaded, and Wanted Languages
Loved
Dreaded
Wanted
Python 25.1%
JavaScript 19.0%
Go 16.2%
Kotlin 12.4%
TypeScript 11.9%
Java 10.5%
C++ 10.2%
Rust 8.3%
C# 8.0%
Swift 7.7%

# PYTHON VS OTHER LANGUAGES




**Python vs. Other Programming Languages: Go, JS, Node.js, Java, Ruby, PHP, R, C++, C#**

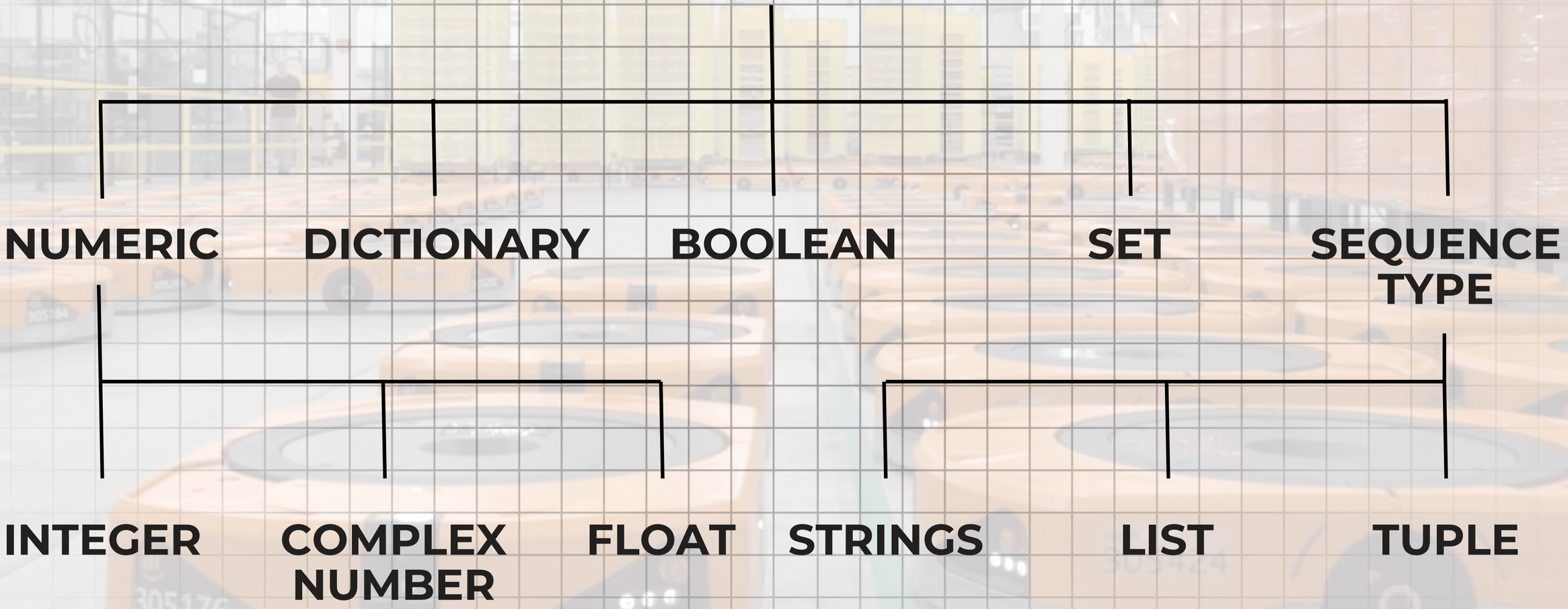
A comparison of Python with other top programming languages. Should you choose Python for your next software project? Read on to find out and decide!

stxnext.com /

# Integrated Development Environment (IDE)

1. PyCharm
2. Jupyter NoteBook
3. Visual Studio Code
4. Google CoLab
5. Spyder
6. PyDev

# Data Types in Python



# Type casting

Method to convert the Python variable datatype into a certain data type in order to perform the required operation by users.

```
a = 7
b = 2

# addition
print ('Sum: ', a + b)

# subtraction
print ('Subtraction: ', a - b)

# multiplication
print ('Multiplication: ', a * b)

# division
print ('Division: ', a / b)

# floor division
print ('Floor Division: ', a // b)

# modulo
print ('Modulo: ', a % b)

# a to the power b
print ('Power: ', a ** b)
```

# OPERATORS

## 1. Arithmetic Operator

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, etc

```
print(21 > 19)

print(18 > 21)

fortnite = 100

pubg = 99

clashofclans = 90


if (fortnite > pubg and fortnite > clashofclans):

    print('Fortnite was the biggest hit of 2018')

else:

    print('Pubg was the biggest hit')


print('a' < 'b')

print('z' > 'y')

print('A' > 'a')

print('The ASCII value of z character is:',ord('z'))
```

# 2. Comparison Operator

Comparison operators in Python, also called relational operators, are used to compare two operands. They return a Boolean True or False depending on whether the comparison condition is true or false

# 3. Logical Operator

```
# Python program to demonstrate
# logical and operator
a = 10
b = 10
c = -10
if a > 0 and b > 0:
    print("The numbers are greater than 0")
if a > 0 and b > 0 and c > 0:
    print("The numbers are greater than 0")
else:
    print("Atleast one number is not greater than 0")
```

AND OPERATOR

```
# Python program to demonstrate
# logical and operator
a = 10
b = 12
c = 0
if a or b or c:
    print("Atleast one number has boolean value as True")
else:
    print("All the numbers have boolean value as False")
```

OR OPERATOR

```
# Python program to demonstrate
# logical not operator
a = 10

if not a:
    print("Boolean value of a is True")
if not (a%3 == 0 or a%5 == 0):
    print("10 is not divisible by either 3 or 5")
else:
    print("10 is divisible by either 3 or 5")
```

NOT OPERATOR

# 4. Assignment Operator

Assignment Operators
are used to assign values to
variables.
They combine the operation
of assigning a value to a variable
with a
mathematical or
bitwise operation.

```
1   x=5
2   x+=5
3   print(x)
4
5   x=2
6   x-=2
7   print(x)
8
9   x=12
10  x*=12
11  print(x)
12
13  x=10
14  x/=5
15  print(x)
16
17  a=4
18  a%=2
19  print(a)
20
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
10
0
144
2.0
0
```

# 5. Identity Operator

```
>>> x = 5
>>> y = 5
>>>
>>> print(x is y)
True
>>>
>>> x = 5
>>> y = 10
>>>
>>> print(x is y)
False
>>>
```

They are used to compare the memory locations of two objects. They determine whether two variables reference the same object in memory.

# 6. Membership Operator

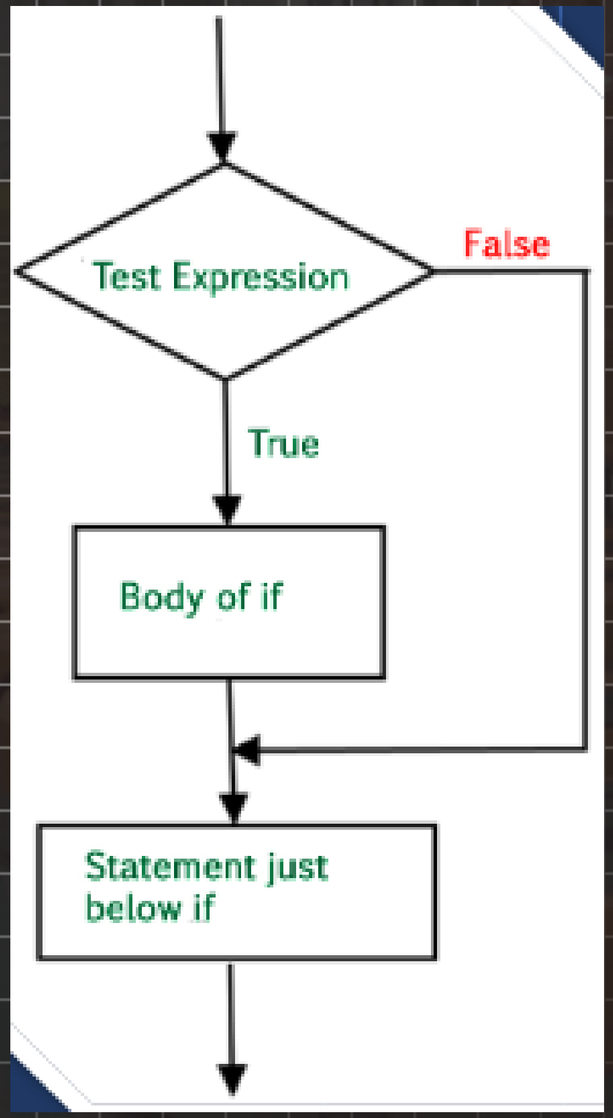
```
x = ["apple", "banana"]
print("banana" in x)

# returns True because a sequence with the value "banana" is in the list

x = ["apple", "banana"]
print("pineapple" not in x)

# returns True because a sequence with the value "pineapple" is not in the list
```

```
True
True
```

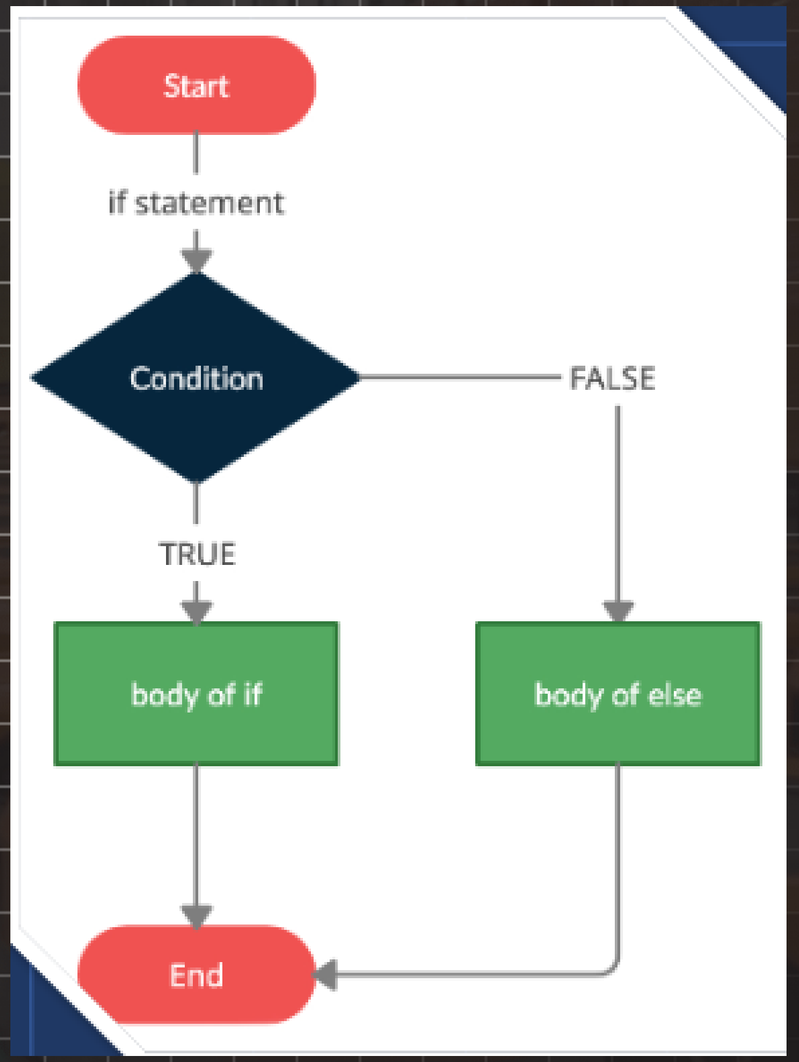
They are used to test whether a value or variable is a member of a sequence

# Conditional Statements

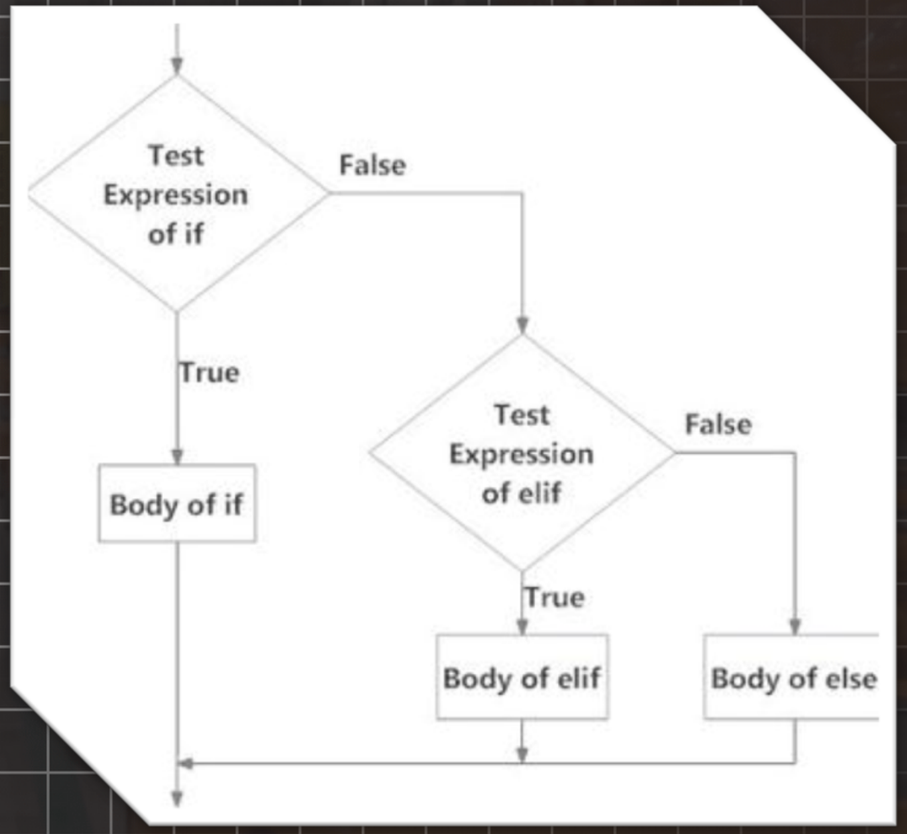
1) If Statement



# 2) If-else Statement



# 3) If-elif-else Statement



# COLLECTIONS

- Containers used for storing data.
- Commonly known as data structures, such as lists, tuples, arrays, dictionaries, etc.
- Python has a built-in collections module providing additional data structures for collections of data.

# Loops in Python

1) For Loop

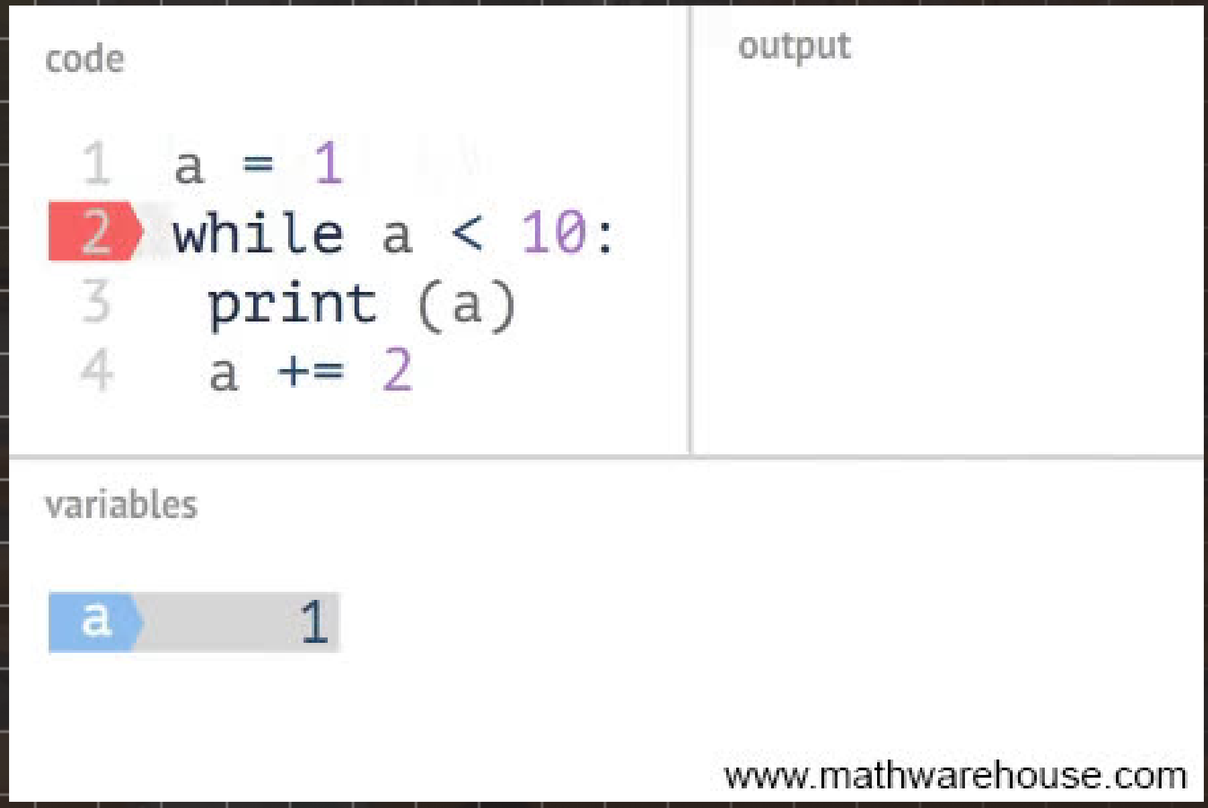
script.py

```
a_list = [1, 3, 5]

for value in a_list:
    print(value)
```

First iteration
value = 1

Output

# 2) While Loop

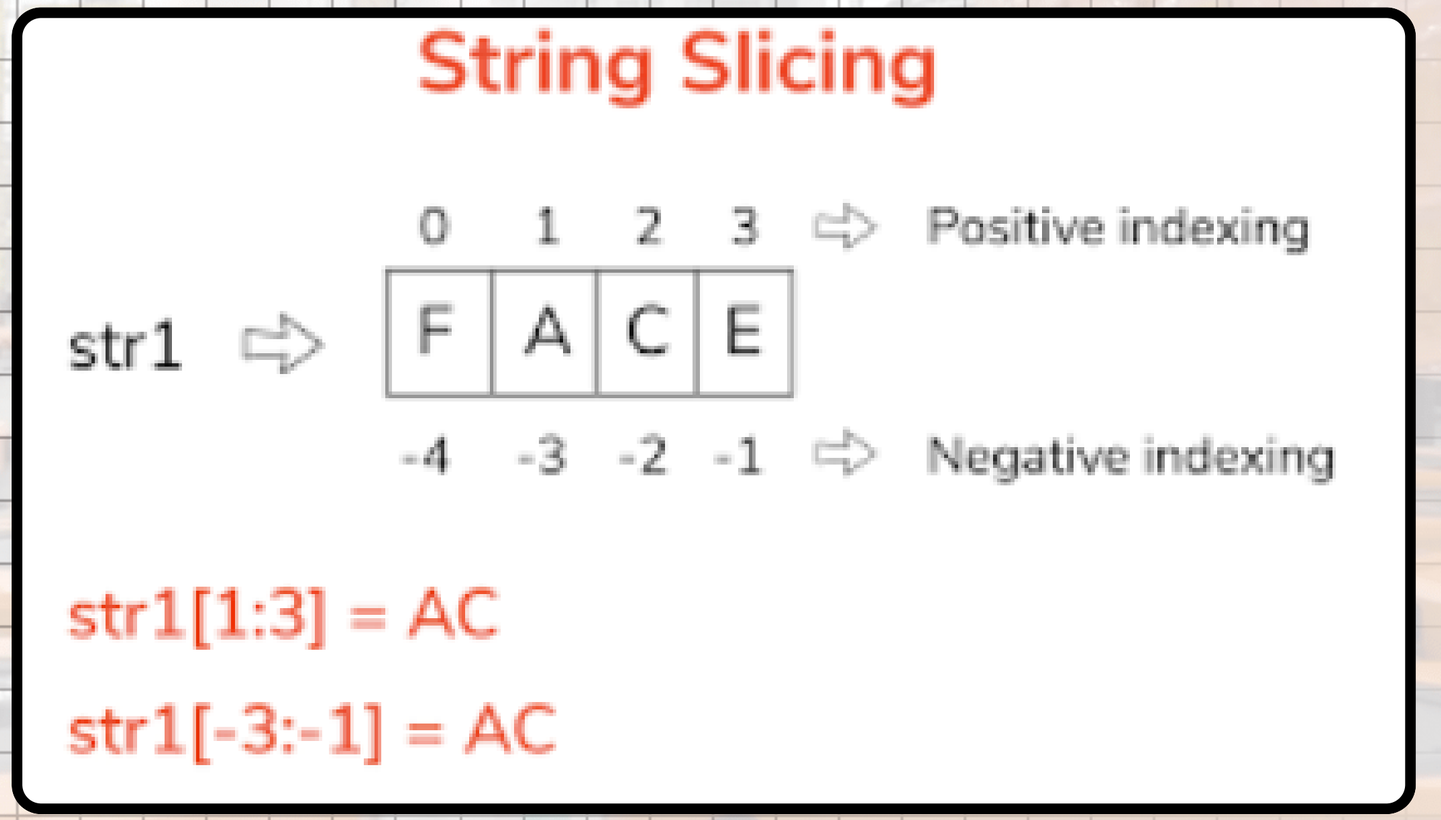

code
1 a = 1
2 while a < 10:
3 print (a)
4 a += 2
output
variables
a 1
www.mathwarehouse.com

# String Manipulation in Python

1) Concatenation

```
Please enter the first string:
Happy
Please enter the second string:
Coding
Concatenated String =
HappyCoding
```

## 2) Slicing


String Slicing
0 1 2 3
Positive indexing
str1
F A C E
-4 -3 -2 -1
Negative indexing
str1[1:3] = AC
str1[-3:-1] = AC

# 3) String Indexing

```
my_string = "Python"
first_char = my_string[0]
third_char = my_string[2]
```

# 3) String Indexing

```
my_string = "Python"
first_char = my_string[0]
third_char = my_string[2]
```

# 4) String Formatting

```
name = "John"

age = 25

message = f"Hello, my name is {name} and I am {age} years old."
```

# 5) String Splitting

```
sentence = "Python is an amazing programming language"

words = sentence.split()
```

# 6) String Joining

```
words = ["Python", "is", "awesome"]
sentence = " ".join(words)
```

# 7) String Length

```
my_string = "Hello, World!"
length = len(my_string)
```

# Classes:

A class is a blueprint for creating objects (instances) with predefined properties and behaviors. It serves as a template for creating objects.

```python
class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def drive(self):
        print(f"{self.brand} {self.model} is driving.")

# Creating an object of the class Car
my_car = Car("Toyota", "Camry")
my_car.drive()  # Output: Toyota Camry is driving.
```

# Objects:

Objects are instances of classes. They encapsulate data for the class and can access the class's methods.

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age


# Creating objects of the class Person
person1 = Person("Alice", 30)
person2 = Person("Bob", 25)
```

# Import Packages:

Import Package:
A package is a collection of Python modules. Importing a package allows you to use its modules and functions in your code.

```
import math

print(math.pi)  # Output: 3.141592653589793
```

**Create Importable Packages:**

- You can create importable packages by organising your code into directories with **__init__.py files** and Python modules.

```
my_package/
    __init__.py
    module1.py
    module2.py
```

# Self Keyword:

The self keyword represents the instance of a class and allows you to access the attributes and methods of the class within its scope.

```python
class Dog:
    def __init__(self, name):
        self.name = name

    def bark(self):
        print(f"{self.name} is barking.")

my_dog = Dog("Buddy")
my_dog.bark()  # Output: Buddy is barking.
```

# With Keyword:

The with keyword is used to wrap the execution of a block of code within a context manager. It simplifies resource management.

```
with open("file.txt", "r") as file:
    data = file.read()
```

# Open Keyword:

The open() function is used to open files in Python. It returns a file object that can be used to read from or write to the file.

```
file = open("file.txt", "r")
data = file.read()
file.close()
```

# Scope of a Variable:

The scope of a variable determines where the variable can be accessed within the program. Variables can have local or global scope.

```
x = 10  # Global variable

def func():
    y = 20  # Local variable
    print(x)  # Accessing global variable

func()
print(y)  # Error: y is not defined
```

# List Comprehension Under Loops::

List comprehension provides a concise way to create lists. It can be used within loops to generate lists based on some conditions.

```
# Example of list comprehension under a loop
numbers = [1, 2, 3, 4, 5]
squared = [x ** 2 for x in numbers]
print(squared)  # Output: [1, 4, 9, 16, 25]
```

# Are the given codes Correct?

1)

```
number = 42
text = "Hello, Python!"
result = number + text
```

2)

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

3)

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

# Did you get them Right?

1) No

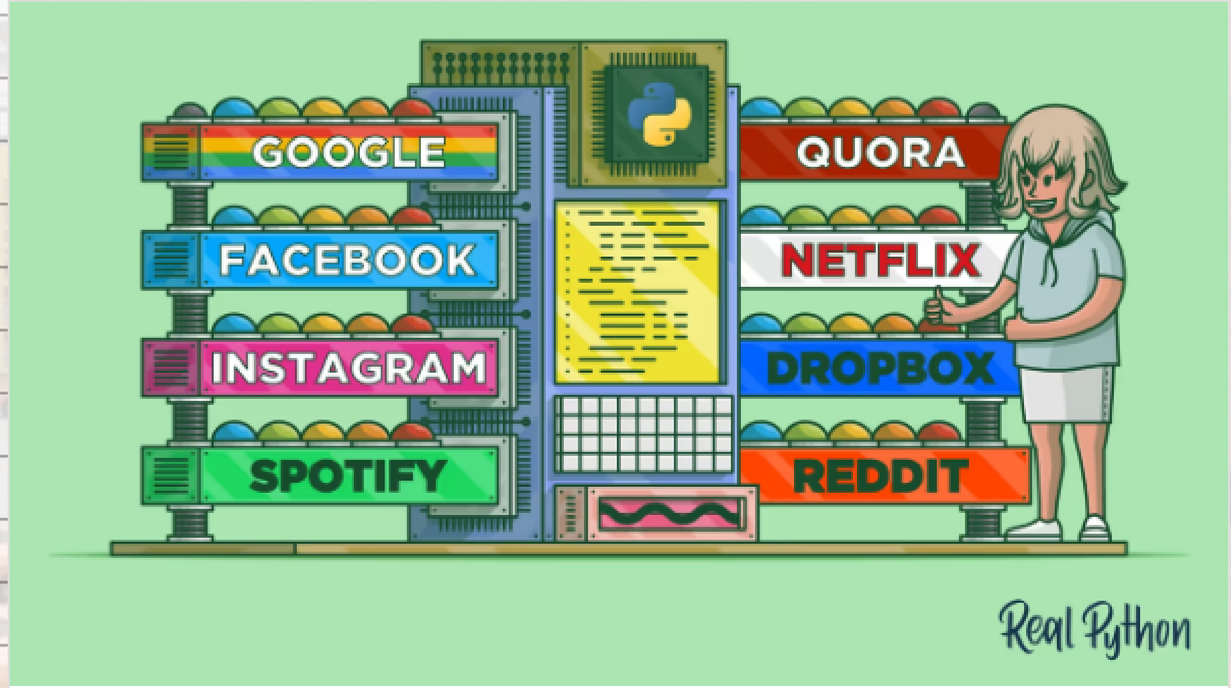
```
number = 42
text = "Hello, Python!"
result = str(number) + text
```

2) No

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

3) Yes!

# WORLD CLASS COMPANIES THAT USE PYTHON




**8 World-Class Software Companies That Use Python**

A review of eight top-tier companies that use Python in production, and why. Learn about their struggles and successes, and see the career opportunities for Python developers today.

realpython /

# Thank You

Any Doubts?