SCHOOL OF ENGINEERING, KMITL

//WK02

Advance Computer Programming

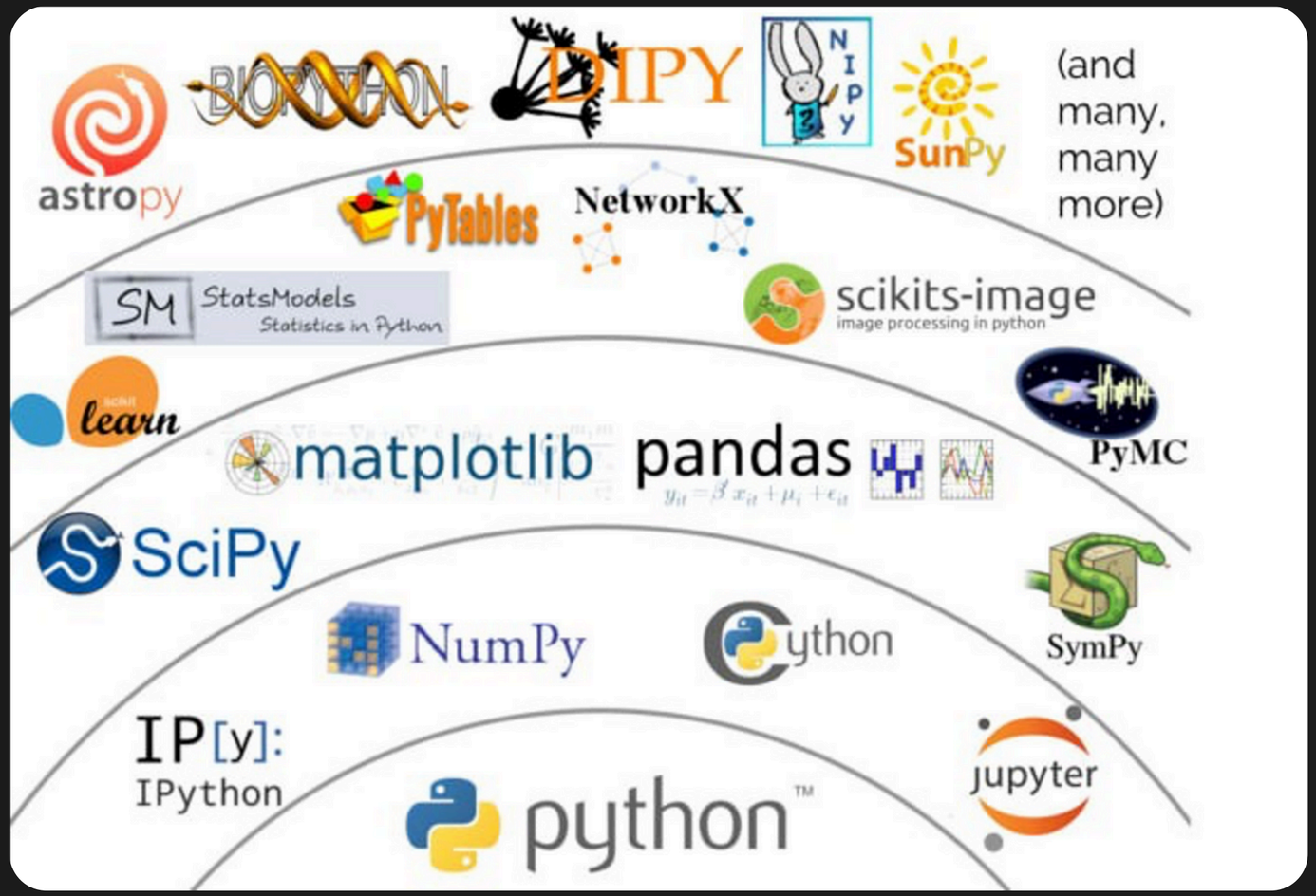
Topic of
Introduction to
Python Programming

Dr. Sarucha Yanyong,
Department of Robotics
and AI Engineering

Advance
Computer
Programming

Why Python?

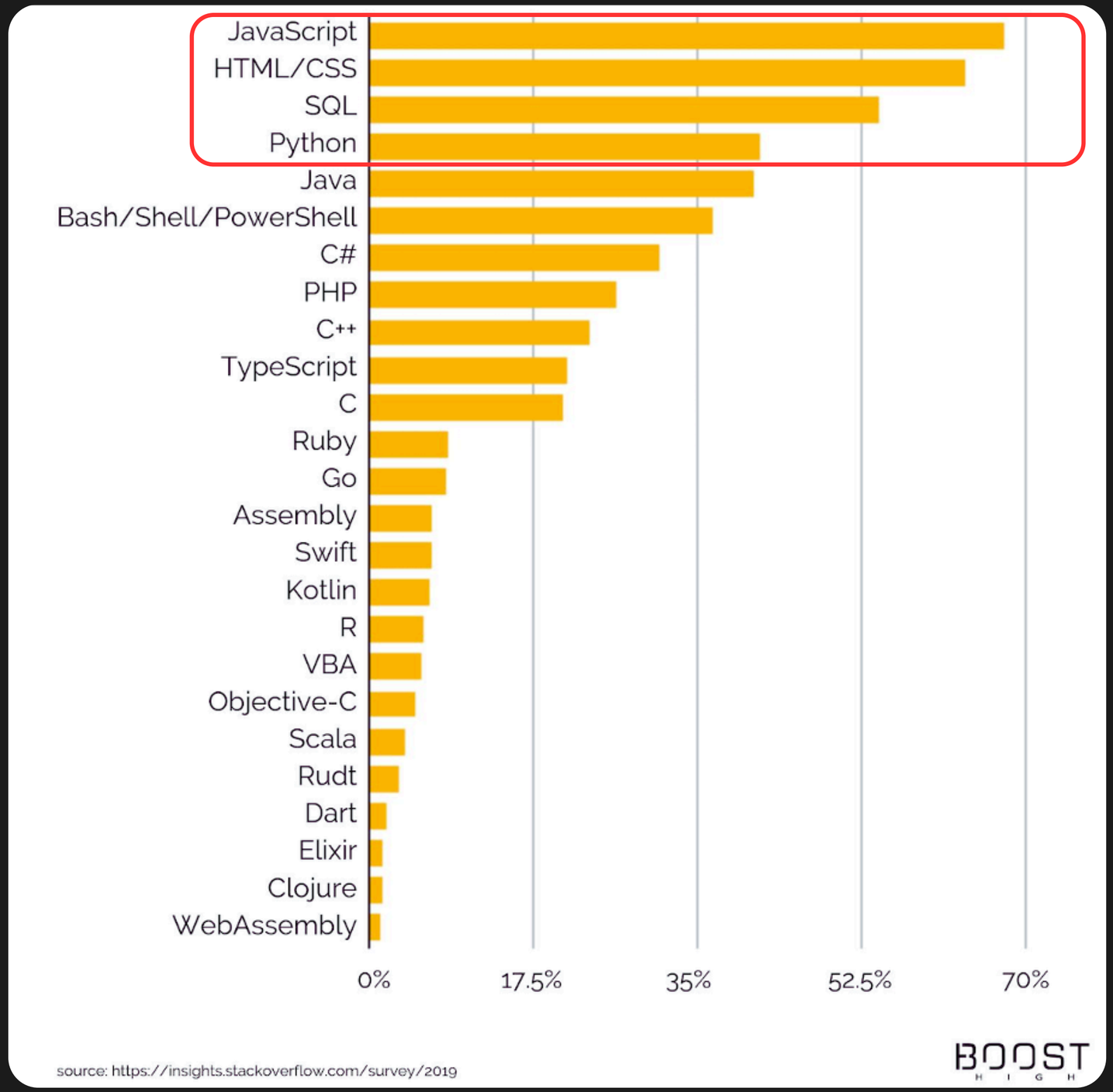
In-term of compatibility



Advance
Computer
Programming

Why Python?

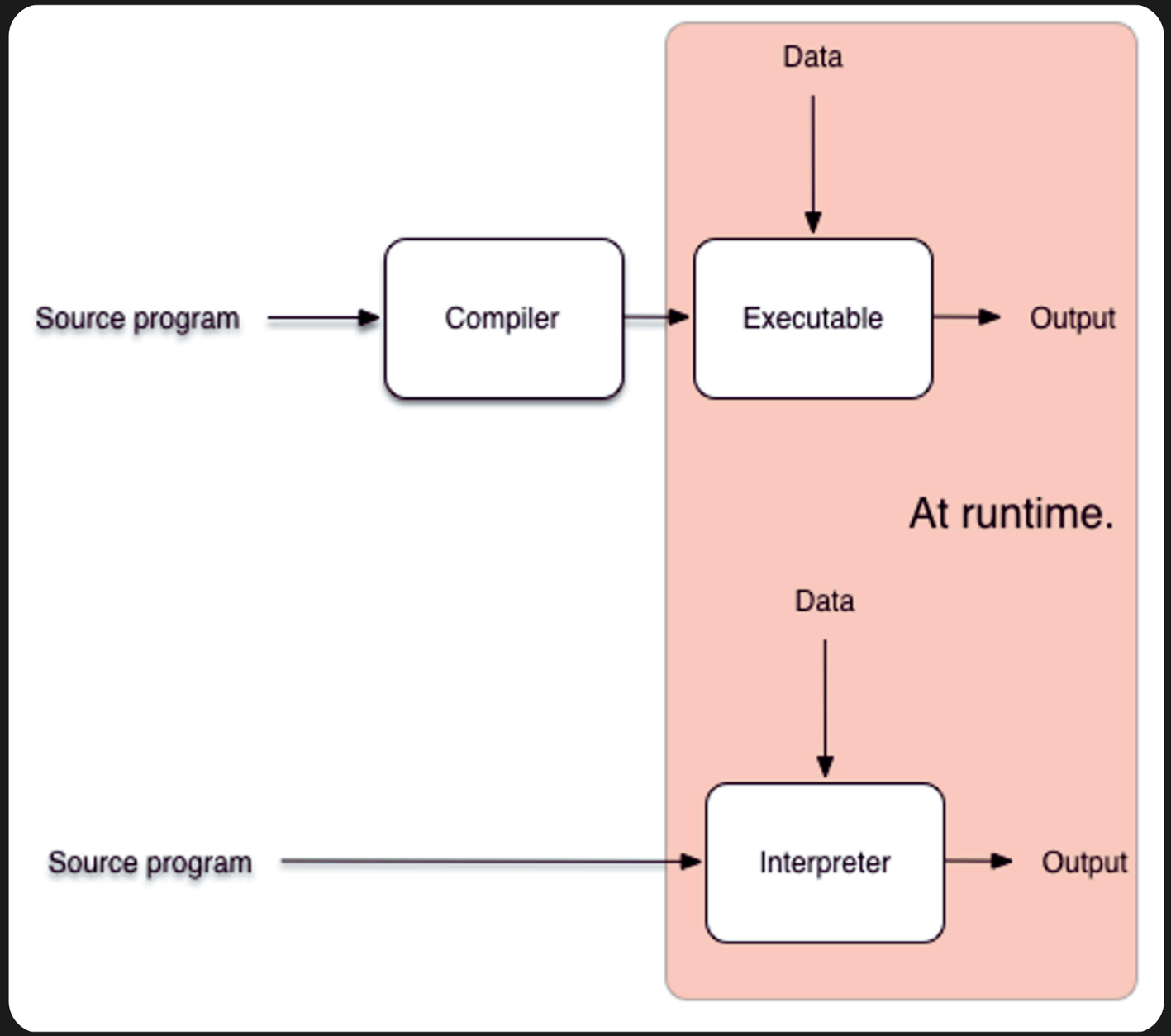
In-term of popularity



Advance
Computer
Programming

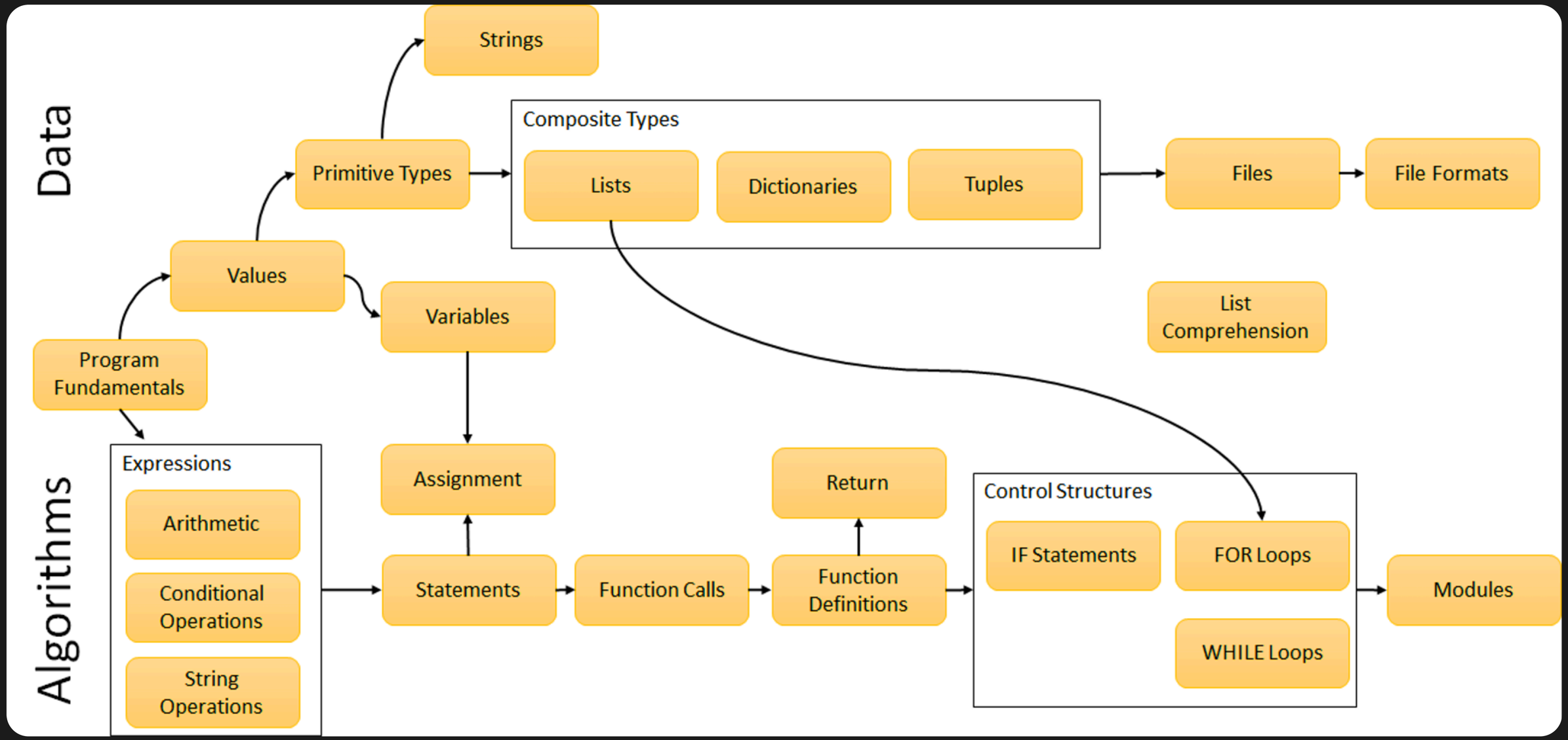
Execution method

Python is an Interpreted Language



Advance
Computer
Programming

Component of Python



Advance
Computer
Programming

Data Types

In Python

Numeric Types

- int: Integer numbers

```
num = 10
```

- float: Floating-point numbers

```
dec = 15.2
```

Text Types

- str: String of characters

```
s = "apple"
```

Sequence Types

- list: Ordered, mutable collection

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

- tuple: Ordered, immutable collection

```
coordinates = (10.0, 20.0)
```

Advance
Computer
Programming

Data Types (2)

In Python

Set Types

- set: Unordered collection of unique elements

```
unique_numbers = {1, 2, 3, 4, 5}
```

Dictionary Types

- dict: Key-value pairs

```
student = {"name": "Alice", "age": 21, "grade": "A"}
```

Boolean Types

- bool: Boolean values (True or False)

```
is_active = True
```

None Type

- NoneType: Represents the absence of a value (None)

```
result = None
```

Advance
Computer
Programming

Using type()

```
# Integer
age = 25
print(type(age)) # Output: <class 'int'>

# Float
price = 19.99
print(type(price)) # Output: <class 'float'>

# List
fruits = ["apple", "banana", "cherry"]
print(type(fruits))  # Output: <class 'list'>
```

In Python, you can use the type() function to determine the type of an object.

Advance
Computer
Programming

string

Strings are ordered sequences, so these features are Indexing, Slicing, Concatenation and Method manipulation.

Advance
Computer
Programming

string

Indexing

```
greeting = "Hello, World!"

# Accessing the first character
first_char = greeting[0]
print(first_char)  # Output: H

# Accessing the fifth character
fifth_char = greeting[4]
print(fifth_char)  # Output: o

# Accessing the last char using negative indexing
last_char = greeting[-1]
print(last_char)  # Output: !
```

Advance
Computer
Programming

string

Slicing

```
greeting = "Hello, World!"

# Slicing the first five characters
first_five = greeting[0:5]
print(first_five)  # Output: Hello

# Slicing from the seventh character to the end
from_seven_to_end = greeting[7:]
print(from_seven_to_end)  # Output: World!

# Slicing with negative indices
slice_negative = greeting[-6:-1]
print(slice_negative)  # Output: World
```

string

upper() method

```
# Original string
greeting = "Hello, World!"

# Convert to uppercase
uppercase_greeting = greeting.upper()

# Display the result
print(uppercase_greeting)  # Output: HELLO, WORLD!
```

The upper() method in Python converts all characters in a string to uppercase.

string

lower() method

```
# Original string
greeting = "Hello, World!"

# Convert to lowercase
lowercase_greeting = greeting.lower()

# Display the result
print(lowercase_greeting)  # Output: hello, world!
```

The lower() method in Python converts all characters in a string to lowercase.

Advance
Computer
Programming
for RAI

string

In Python, you can add numbers into strings using different methods such as string formatting with the % operator, str.format(), and f-strings (formatted string literals).

Advance
Computer
Programming
for RAI

string

formating using
the % Operator

```
x = "No. %d" % (100)
print(x)  # Output: No. 100
```

string

formating using
str.format()

```
x = "No. {}".format(100)
print(x)  # Output: No. 100
```

string

Using f-Strings (Formatted String Literals)

```
x = f"No. {100}"
print(x)  # Output: No. 100
```

Advance
Computer
Programming
for RAI

list

A list in Python is an ordered, mutable collection of elements. It can hold a variety of data types, including numbers, strings, and even other lists.

list

Key Characteristics:

- **Ordered**: The items have a defined order, and that order will not change unless explicitly modified.
- **Mutable**: You can change, add, and remove items after the list has been created.
- **Heterogeneous**: A list can contain items of different data types.

list

Creating

```
# Creating a list of strings
fruits = ["apple", "banana", "cherry"]

# Creating a list of mixed data types
mixed_list = [1, "hello", 3.14, True]

# Creating an empty list
empty_list = []
```

Advance
Computer
Programming
for RAI

list

Common List Operations

Accessing Elements:

```
first_fruit = fruits[0]  # Output: "apple"
```

Modifying Elements:

```
fruits[1] = "blueberry"
# fruits is now ["apple", "blueberry", "cherry"]
```

Adding Elements:

```
fruits.append("orange")  # fruits is now
["apple", "blueberry", "cherry", "orange"]
```

list

Common List Operations (2)

Inserting Elements:

```
fruits.insert(1, "banana")
# fruits is now ["apple", "banana", "blueberry",
"cherry", "orange"]
```

Removing Elements: Removes the first occurrence of a specified value

```
fruits.remove("banana")  # fruits is now
["apple", "blueberry", "cherry", "orange"]
```

Pop Elements: Remove the last element and returns an item

```
last_fruit = fruits.pop()
# Output: "orange", fruits is now ["apple",
"blueberry", "cherry"]
```

list

Common List Operations (3)

Pop Elements: Remove the specific element by index

```
last_fruit = fruits.pop(1)
# Output: "apple", fruits is now ["blueberry",
"cherry"]
```

Slicing: Extract a portion of the list

```
fruits = ["apple", "blueberry", "cherry"]
slice_of_fruits = fruits[1:3]
# Output: ["blueberry", "cherry"]
```

**The stop index in slicing is exclusive, meaning it is not included in the resulting slice.

list

Common List Operations (4)

Length: Get the number of items in the list.

```
length_of_fruits = len(fruits)  # Output: 2
```

Sorting: Sort the list in ascending or descending order.

```
fruits.sort()
# fruits is now ["apple", "blueberry", "cherry"]

fruits.sort(reverse=True)
# fruits is now ["cherry", "blueberry", "apple"]
```

Combining Lists: Concatenate two lists.

```
more_fruits = ["kiwi", "mango"]
all_fruits = fruits + more_fruits
# Output: ["cherry", "blueberry", "apple",
"kiwi", "mango"]
```

list

Common List
Operations (5)

dictionary

An unordered, mutable collection of key-value pairs. Each key is unique, and it maps to a value. Dictionaries are optimized for retrieving values when the key is known

dictionary

Key Characteristics:

- **Unordered**: The items do not have a defined order.
- **Mutable**: You can change, add, or remove items.
- **Unique Keys**: Each key must be unique.
- **Key-Value Pairs**: Each key is associated with a value.

dictionary

Syntax

```
dictionary_name = {
    key1: value1,
    key2: value2,
    key3: value3,
    ...
}
```

dictionary

Example

```
student = {
    "name": "Alice",
    "age": 21,
    "grade": "A",
    "courses": ["Math", "Science", "English"]
}
```

Advance
Computer
Programming
for RAI

dictionary

Accessing Values: Using keys to access the values.

```
name = student["name"]
print(name)   # Output: Alice
```

Modifying Values: Changing the value associated with a key.

```
student["age"] = 22
print(student["age"])   # Output: 22
```

Adding: Adding a new key-value pair to the dictionary

```
student["major"] = "RAI"
print(student["major"]) # Output: RAI
```

dictionary

Removing Items: Using del to remove a key-value pair.

```
del student["grade"]
print(student)
# Output: {'name': 'Alice', 'age': 22,
'courses': ['Math', 'Science', 'English'],
'major': 'Computer Science'}
```

Removing Items: Using pop() to remove a key-value pair and return the value.

```
age = student.pop("age")
print(age)  # Output: 22
print(student)  # Output: {'name':
'Alice', 'courses': ['Math', 'Science',
'English'], 'major': 'Computer Science'}
```

dictionary

Method

get(): Returns the value for a key, or a default value if the key is not found (But show error).

```
grade = student.get("grade", "No grade")
print(grade)  # Output: No grade
```

1

```
grade = student.get("grade")
print(grade)  # Output: None
```

2

keys(): Returns a view object containing the keys of the dictionary.

```
keys = student.keys()
print(keys)
# Output: dict_keys(['name', 'courses',
'major'])
```

Advance
Computer
Programming
for RAI

dictionary

Method

values(): Returns a view object containing the values of the dictionary.

```
values = student.values()
print(values)
# Output: dict_values(['Alice', ['Math',
'Science', 'English'], 'Computer
Science'])
```

update(): Updates the dictionary with key-value pairs from another dictionary or an iterable of key-value pairs.

```
student.update({"grade": "A", "age": 22})
print(student)
# Output: {'name': 'Alice', 'courses':
['Math', 'Science', 'English'], 'major':
'Computer Science', 'grade': 'A', 'age':
22}
```

Advance
Computer
Programming
for RAI

dictionary

Iterating

Looping through keys, values, or key-value pairs.

```
for key in student:
    print(key, student[key])

# Output:
# name Alice
# courses ['Math', 'Science', 'English']
# major Computer Science
# grade A
# age 22
```

dictionary

Iterating (2)

Looping through keys, values, or key-value pairs using items.

```
for key, value in student.items():
    print(f"{key}: {value}")

# Output:
name: Alice
age: 21
grade: A
courses: ['Math', 'Science', 'English']
```

Conditions

if

Syntax of an **if** statement

```
if some_condition:
    # execute some code
```

```
if 3>2:
    print('ITS TRUE!')
```

Conditions

if..else

Syntax of an **if/else** statement

```
if some_condition:
    # execute some code
else:
    # do something else
```

```
hungry = True

if hungry==True:
    print('FEED ME!')
else:
    print("Im not hungry")
```

```
FEED ME!
```

Advance
Computer
Programming
for RAI

Conditions

if...elif...else

```
loc = 'Bank'

if loc == 'Auto Shop':
    print("Cars are cool!")
elif loc == 'Bank':
    print("Money is cool!")
elif loc == 'Store':
    print("Welcome to the store!")
else:
    print("I do not know much.")
```

```
Money is cool!
```

Conditions

Loops: for

Syntax of a for loop

```
my_iterable = [1,2,3]
for item_name in my_iterable:
    print(item_name)
```

```
for i in range(0, 5):
    print(i)
```

Conditions

Loops: while

Syntax of a while loop

```
while some_boolean_condition:
    #do something
```

```
i = 0
while i < 5:
    print(f"No. is {i}")
    i = i + 1
```

```
No. is 0
No. is 1
No. is 2
No. is 3
No. is 4
```

Conditions

Loops: while

Syntax of a while loop

```
while some_boolean_condition:
    #do something
```

```
i = 0
while i < 5:
    print(f"No. is {i}")
    i = i + 1
```

```
No. is 0
No. is 1
No. is 2
No. is 3
No. is 4
```

Function vs Method

A function is an independent block of code, while a method is a function associated with an **object** and defined within a **class**.

Function

def

```
def name_of_function():
    '''
    Docstring explains function.
    '''
    print("Hello")
```

Advance
Computer
Programming
for RAI

Function

def

```
# Try 1
def my_function1():
    print("hello world")
print(my_function1())

# Try 2
def my_function2():
    return "hello world"
print(my_function2())
```

Advance
Computer
Programming
for RAI

Module

A module in Python is a single file containing Python code. It can include functions, classes, and variables, as well as runnable code. Modules help in organizing and reusing code across different programs.

Advance
Computer
Programming
for RAI

Module

Creating your own library as module in python

(1) Create a file named **mymodule.py** with a function definition.

```
# mymodule.py
def greet(name):
  return f"Hello, {name}!"
```

(2) Create the main script file named **main.py** that will import and use the function from mymodule.py.

```
# main.py
from mymodule import greet


# Call the function and print the result
greeting = greet("Alice")
print(greeting)  # Output: Hello, Alice!
```