Day2 of Internship

- **Introduction**

This documentation covers the essentials of Python programming, coding standards, and APIs. It explains Python basics like data types, loops, functions, and OOP, while also highlighting clean coding practices such as PEP8 and SOLID principles. The API section introduces concepts like CRUD operations, authentication, and status codes, along with practical use in Python. Extra topics like SDLC, Agile, version control, and software architecture link everything to real-world software development.

- **Research Topics**

Python Syntax

Python syntax is like grammar for this programming language. Syntax refers to the set of rules that defines how to write and organize code so that the Python interpreter can understand and run it correctly. These rules ensure that your code is structured, formatted, and error-free. Here are some basic Python syntax:

Python Indentation refers to the use of whitespace (spaces or tabs) at the beginning of code line. It is used to define the code blocks. Indentation is crucial in Python because, unlike many other programming languages that use braces "{}" to define blocks, Python uses indentation. It improves the readability of Python code, but on other hand it became difficult to rectify indentation errors. Even one extra or less space can lead to indentation error.

```
if 10 > 5:
    print("This is true!")
    print("I am tab indentation")

print("I have no indentation")
```

Variables in Python are essentially named references pointing to objects in memory. Unlike some other languages, you don't need to declare a variable's type explicitly in Python. Based on the value assigned, Python will dynamically determine the type.

In the below example, variable 'a' is initialize with integer value and variable 'b' with a string. Because of dynamic-types behaviour, data type will be decide during runtime.

```
a = 10

print(type(a))


b = 'GeeksforGeeks'

print(type(b))
```

In Python, identifiers are unique names that are assigned to variables, functions, classes, and other entities. They are used to uniquely identify the entity within the program. They should start with a letter (a-z, A-Z) or an underscore "_" and can be followed by letters, numbers, or underscores.In the below example "first_name" is an identifier that store string value.

```
first_name = "Ram"
```

For naming of an identifier we have to follows some rules given below:

Identifiers can be composed of alphabets (either uppercase or lowercase), numbers (0-9), and the underscore character (_). They shouldn't include any special characters or spaces.

The starting character of an identifier must be an alphabet or an underscore.

Within a specific scope or namespace, each identifier should have a distinct name to avoid conflicts. However, different scopes can have identifiers with the same name without interference.

Python keywords

Keywords in Python are reserved words that have special meanings and serve specific purposes in the language syntax. They cannot be used as identifiers (names for variables, functions, classes, etc.). For instance, "for", "while", "if", and "else" are keywords and cannot be used as identifiers.

Below is the list of keywords in Python:

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Comments in Python are statements written within the code. They are meant to explain, clarify, or give context about specific parts of the code. The purpose of comments is to explain the working of a code, they have no impact on the execution or outcome of a program.

Single line comments are preceded by the "#" symbol. Everything after this symbol on the same line is considered a comment.

```
first_name = "Reddy"

last_name = "Anna"  # assign last name

# print full name

print(first_name, last_name)
```

Output

```
Reddy Anna
```

multi-line comments: Python doesn't have a specific syntax for multi-line comments. However, programmers often use multiple single-line comments, one after the other, or sometimes triple quotes (either ''' or """), even though they're technically string literals. Below is the example of multiline comment.

```
'''

Multi Line comment.

Code will print name.

'''

f_name = "Alen"

print(f_name)
```

Multiple Line Statements: Writing a long statement in a code is not feasible or readable. Breaking a long line of code into multiple lines makes is more readable.

Using Backslashes (\): In Python, you can break a statement into multiple lines using the backslash (\). This method is useful, especially when we are working with strings or mathematical operations.

```
sentence = "This is a very long sentence that we want to " \

        "split over multiple lines for better readability."


print(sentence)

# For mathematical operations
```

```
total = 1 + 2 + 3 + \
        4 + 5 + 6 + \
        7 + 8 + 9
print(total)
```

Output

```
This is a very long sentence that we want to split over multiple lines for better readability.
45
```

Continuation of Statements in Python: In Python, statements are typically written on a single line. However, there are scenarios where writing a statement on multiple lines can improve readability or is required due to the length of the statement. This continuation of statements over multiple lines is supported in Python in various ways:

Implicit Continuation: Python implicitly supports line continuation within parentheses (), square brackets [], and curly braces {}. This is often used in defining multi-line lists, tuples, dictionaries, or function arguments.

```
# Line continuation within square brackets '[]'
numbers = [
    1, 2, 3,
    4, 5, 6,
    7, 8, 9
]
print(numbers)
```

Explicit Continuation: You can use backslash '\' to indicate that a statement should continue on the next line.

```
# Explicit continuation
s = "GFG is computer science portal " \
    "by Geeks, used by Geeks."
print(s)
```

Note: Using a backslash does have some pitfalls, such as if there's a space after the backslash, it will result in a syntax error.

The input() function in Python is used to take user input from the console. The program execution halts until the user provides input and presses "Enter". The entered data is then

returned as a string. We can also provide an optional prompt as an argument to guide the user on what to input.

Example: In this example, the user will see the message "Please enter your name: ". After entering their name and pressing "Enter", they'll receive a greeting with the name they provided.

```
# Taking input from the user

name = input("Please enter your name: ")

# Print the input

print(f"Hello," + name)
```

Output:

```
Please enter your name:

Peter

Hello, Peter!
```

Source: https://www.geeksforgeeks.org/python/python-syntax/

Variables

In Python, variables are used to store data that can be referenced and manipulated during program execution. A variable is essentially a name that is assigned to a value. Unlike many other programming languages, Python variables do not require explicit declaration of type. The type of the variable is inferred based on the value assigned.Variables act as placeholders for data. They allow us to store and reuse values in our program.

```
# Variable 'x' stores the integer value 10

x = 5

# Variable 'name' stores the string "Samantha"

name = "Samantha"

print(x)

print(name)
```

Output

```
5

Samantha
```

Rules for Naming Variables

To use variables effectively, we must follow Python's naming rules:

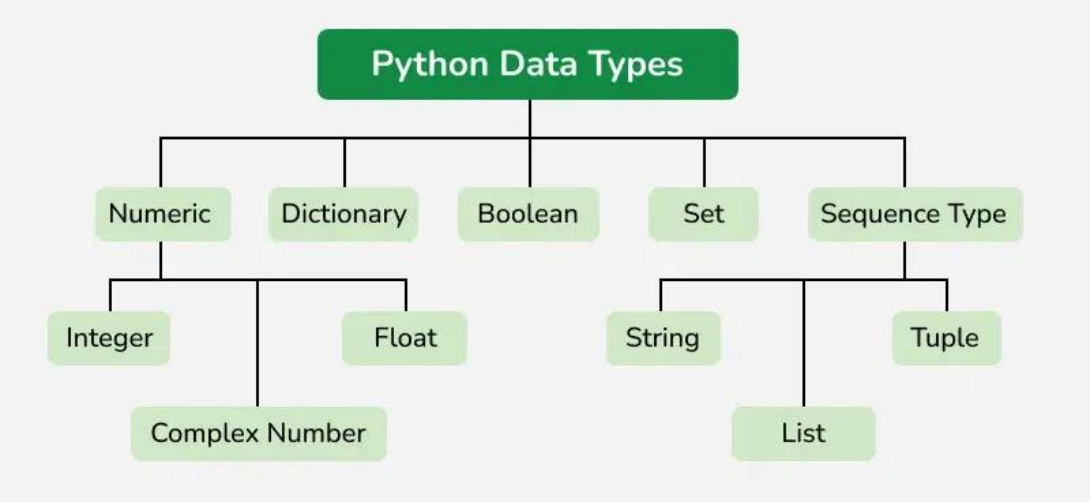
- Variable names can only contain letters, digits and underscores (_).
- A variable name cannot start with a digit.
- Variable names are case-sensitive (myVar and myvar are different).
- Avoid using Python keywords (e.g., if, else, for) as variable names.

Source: https://www.geeksforgeeks.org/python/python-variables/

Datatypes

Data types in Python are a way to classify data items. They represent the kind of value, which determines what operations can be performed on that data. Since everything is an object in Python programming, Python data types are classes and variables are instances (objects) of these classes.

The following are standard or built-in data types in Python:



1. Numeric Data Types

Python numbers represent data that has a numeric value. A numeric value can be an integer, a floating number or even a complex number. These values are defined as int, float and complex classes.

- **Integers:** value is represented by int class. It contains positive or negative whole numbers (without fractions or decimals). There is no limit to how long an integer value can be.
- **Float:** value is represented by float class. It is a real number with a floating-point representation. It is specified by a decimal point. Optionally, character e or E followed by a positive or negative integer may be appended to specify scientific notation.

- **Complex Numbers:** It is represented by a complex class. It is specified as (real part) + (imaginary part)j. For example - 2+3j

```
a = 5

print(type(a))

b = 5.0

print(type(b))

c = 2 + 4j

print(type(c))
```

Output

```
<class 'int'>

<class 'float'>

<class 'complex'>
```

2. Sequence Data Types

A sequence is an ordered collection of items, which can be of similar or different data types. Sequences allow storing of multiple values in an organized and efficient fashion. There are several sequence data types of Python:

String

Python Strings are arrays of bytes representing Unicode characters. In Python, there is no character data type, a character is a string of length one. It is represented by str class.

Strings in Python can be created using single quotes, double quotes or even triple quotes. We can access individual characters of a String using index.

```
s = 'Welcome to the Geeks World'

print(s)

# check data type

print(type(s))

# access string with index

print(s[1])

print(s[2])

print(s[-1])
```

Output

```
Welcome to the Geeks World
<class 'str'>
e
l
d
```

List

Lists are similar to arrays found in other languages. They are an ordered and mutable collection of items. It is very flexible as items in a list do not need to be of the same type.

Creating a List in Python

Creating a List in Python

Lists in Python can be created by just placing sequence inside the square brackets[].

```
# Empty list
a = []
# list with int values
a = [1, 2, 3]
print(a)
# list with mixed values int and String
b = ["Geeks", "For", "Geeks", 4, 5]
print(b)
```

Output

```
[1, 2, 3]
['Geeks', 'For', 'Geeks', 4, 5]
```

Access List Items

In order to access the list items refer to index number. In Python, negative sequence indexes represent positions from end of the array. Instead of having to compute offset as in List[len(List)-3], it is enough to just write List[-3]. Negative indexing means beginning from end, -1 refers to last item, -2 refers to second-last item, etc.

```
a = ["Geeks", "For", "Geeks"]
print("Accessing element from the list")
```

```
print(a[0])

print(a[2])

print("Accessing element using negative indexing")

print(a[-1])

print(a[-3])
```

Output

```
Accessing element from the list

Geeks

Geeks

Accessing element using negative indexing

Geeks

Geeks
```

Tuple

Tuple is an ordered collection of Python objects. The only difference between a tuple and a list is that tuples are immutable. Tuples cannot be modified after it is created.

Creating a Tuple in Python

In Python, tuples are created by placing a sequence of values separated by a 'comma' with or without the use of parentheses for grouping data sequence. Tuples can contain any number of elements and of any datatype (like strings, integers, lists, etc.).

```
# initiate empty tuple

tup1 = ()

tup2 = ('Geeks', 'For')

print("\nTuple with the use of String: ", tup2)
```

Output

```
Tuple with the use of String:  ('Geeks', 'For')
```

Access Tuple Items

In order to access tuple items refer to the index number. Use the index operator [ ] to access an item in a tuple.

```
tup1 = tuple([1, 2, 3, 4, 5])
```

```
# access tuple items

print(tup1[0])

print(tup1[-1])

print(tup1[-3])
```

Output

```
1

5

3
```

3. Boolean

Python Boolean Data type is one of the two built-in values, True or False. Boolean objects that are equal to True are truthy (true) and those equal to False are falsy (false). However non-Boolean objects can be evaluated in a Boolean context as well and determined to be true or false. It is denoted by class bool.

```
print(type(True))

print(type(False))

print(type(true))
```

Output:

```
<class'bool'>
<class 'bool'>
```

4. Set Data Type in Python

In Python Data Types, Set is an unordered collection of data types that is iterable, mutable, and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements.

Create a Set in Python

Sets can be created by using the built-in set() function with an iterable object or a sequence by placing the sequence inside curly braces, separated by a ‘comma’. The type of elements in a set need not be the same, various mixed-up data type values can also be passed to the set.

```
# initializing empty set

s1 = set()

s1 = set("GeeksForGeeks")

print("Set with the use of String: ", s1)
```

```
s2 = set(["Geeks", "For", "Geeks"])

print("Set with the use of List: ", s2)
```

Output

```
Set with the use of String:  {'s', 'o', 'F', 'G', 'e', 'k', 'r'}

Set with the use of List:  {'Geeks', 'For'}
```

Access Set Items

Set items cannot be accessed by referring to an index, since sets are unordered the items have no index. But we can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the keyword in.

```
set1 = set(["Geeks", "For", "Geeks"]) #Duplicates are removed automatically

print(set1)

# loop through set

for i in set1:

  print(i, end=" ") #prints elements one by one

 # check if item exist in set

print("Geeks" in set1)
```

Output

```
{'For', 'Geeks'}

For Geeks True
```

5. Dictionary Data Type

A dictionary in Python is a collection of data values, used to store data values like a map, unlike other Python Data Types, a Dictionary holds a key: value pair. Key-value is provided in dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon : , whereas each key is separated by a 'comma'.

Create a Dictionary in Python

Values in a dictionary can be of any datatype and can be duplicated, whereas keys can't be repeated and must be immutable. The dictionary can also be created by the built-in function **dict().**

```
# initialize empty dictionary

d = {}
```

```
d = {1: 'Geeks', 2: 'For', 3: 'Geeks'}

print(d)

# creating dictionary using dict() constructor

d1 = dict({1: 'Geeks', 2: 'For', 3: 'Geeks'})

print(d1)
```

Output

```
{1: 'Geeks', 2: 'For', 3: 'Geeks'}

{1: 'Geeks', 2: 'For', 3: 'Geeks'}
```

Accessing Key-value in Dictionary

In order to access items of a dictionary refer to its key name. Key can be used inside square brackets. Using get() method we can access dictionary elements.

```
d = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}

# Accessing an element using key

print(d['name'])

# Accessing a element using get

print(d.get(3))
```

Output

```
For

Geeks
```

Source: https://www.geeksforgeeks.org/python/python-data-types/

Conditional statements (if-elif-else)

In Python, conditional statements help control the flow of a program by executing different blocks of code based on whether a condition is true or false. These statements allow decision-making in code. The main types of conditional statements are:

- if
- if-else
- nested if
- if-elif

if Statement

An if statement is used when a block of code needs to be executed only if a specific condition evaluates to True.

Syntax

```
if condition:
        #statement to execute if condition is true
```

```
if 10 > 5:
   print("10 greater than 5")
print("Program ended")
```

Output

```
10 greater than 5
Program ended
```

if else Statement

In conditional if Statement the additional block of code is merged as else statement which is performed when if condition is false.

Syntax

```
if (condition):
        #execute this block if condition is true
else:
        #execute this block if condition is false
```

```
x = 3
if x == 4:
   print("Yes")
else:
   print("No")
```

Output

```
No
```

Nested if Statement

if statement can also be checked inside other if statement. This conditional statement is called a nested if statement. This means inner if condition will only be checked if the outer if condition is True.

Syntax

```
 if (condition1):
        #execute if condition1 is true
if(condition1):
        #execute if condition2 is true
```

```
a = 10
if a > 5:
   print("Bigger than 5")
   if a <= 15:
      print("Between 5 and 15")
```

Output

```
Bigger than 5
Between 5 and 15
```

if-elif Statement

The if-elif statement is a shortcut for chaining multiple if-else conditions. While using if-elif statement at the end else block is added which is performed if none of the above if-elif statement is true.

Syntax

```
if(condition):
        statement
elif(condition):
        statement
else:
        statement
```

Source:https://www.geeksforgeeks.org/python/python3-if-if-else-nested-if-if-elif-statements/

Loops (for, while, break, continue)

for loop

Python **for loops** are used for iterating over sequences like lists, tuples, strings and ranges.

- A for loop allows you to apply the same operation to every item within the loop.
- Using a for loop avoids the need to manually manage the index.
- A for loop can iterate over any iterable object, such as a dictionary, list or custom iterator.

```
s = ["Geeks", "for", "Geeks"]

# using for loop with string

for i in s:

    print(i)
```

Output

```
Geeks

for

Geeks
```

while loop

Python While Loop is used to execute a block of statements repeatedly until a given condition is satisfied. When the condition becomes false, the line immediately after the loop in the program is executed.

In this example, the condition for while will be True as long as the counter variable (count) is less than 3.

```
count = 0

while count < 3:

    count = count + 1

    print("Hello Geek")
```

Output

```
Hello Geek

Hello Geek

Hello Geek
```

Break

The break statement in Python is used to exit or "break" out of a loop (either a for or while loop) prematurely, before the loop has iterated through all its items or reached its condition. When the break statement is executed, the program immediately exits the loop, and the control moves to the next line of code after the loop.

```
# Example: Searching for an element in a list
a = [1, 3, 5, 7, 9, 11]
val = 7
for i in a:
    if i == val:
        print(f"Found at {i}!")
        break
else:
    print(f"not found")
```

Output

```
Found at 7!
```

Continue

Python continue statement is a loop control statement that forces to execute the next iteration of the loop while skipping the rest of the code inside the loop for the current iteration only, i.e. when the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped for the current iteration and the next iteration of the loop will begin.

```
for i in range(1, 11):
    if i == 6:
        continue
    print(i, end=" ")
```

Output

```
1 2 3 4 5 7 8 9 10
```

Source: https://www.geeksforgeeks.org/python/python-continue-statement/

Functions (parameters, return values, default args, *args, **kwargs)

Parameters

Parameters are variables defined in a function declaration. This act as placeholders for the values (arguments) that will be passed to the function.

A parameter is the variable defined within the parentheses when we declare a function.

```
# Here a,b are the parameters
def sum(a,b):
  print(a+b)
  sum(1,2)
```

Output

```
3
```

Return values

A return statement is used to end the execution of the function call and it "returns" the value of the expression following the return keyword to the caller.

```
def add(a, b):
 # returning sum of a and b
    return a + b
def is_true(a)
 # returning boolean of a
    return bool(a)
# calling function
res = add(2, 3)
print(res)
res = is_true(2<5)
print(res)
```

Output

```
5
True
```

Argument

An argument is a value that is passed to a function when it is called. It might be a variable, value or object passed to a function or method as input.

```
def sum(a,b):
  print(a+b)
  # Here the values 1,2 are arguments
sum(1,2)
```

Output

```
3
```

*args

The special syntax ***args** allows us to pass any number of positional (non-keyword) arguments to a function.

```
def myFun(*argv):
    for arg in argv:
        print(arg)
myFun('Hello', 'Welcome', 'to', 'GeeksforGeeks')
```

Output

```
Hello
Welcome
To
GeeksforGeeks
```

**kwargs

The special syntax ****kwargs** allows us to pass any number of keyword arguments (arguments in the form key=value). These arguments are collected into a **dictionary**, where:

- Keys = argument names
- Values = argument values

```
def fun(**kwargs):
    for k, val in kwargs.items():
        print(k, "=", val)
fun(s1='Python', s2='is', s3='Awesome')
```

Output

```
s1 = Python
s2 = is
s3 = Awesome
```

Exception Handling (try-except-finally)

The try...except statement allows you to catch one or more exceptions in the try clause and handle each of them in the except clauses.

The try...except statement also has an optional clause called finally:

```
a = 10
b = 0
try:
    c = a / b
    print(c)
except ZeroDivisionError as error:
    print(error)
finally:
    print('Finishing up.')
```

Code language: PHP (php)

Output:

```
division by zero
Finishing up.
```

In this example, the try clause causes a ZeroDivisionError exception both except and finally clause executes.

The try clause in the following example doesn't cause an error. Therefore, all statements in the try and finally clauses execute:

```
a = 10
b = 2
try:
    c = a / b
```

```
  print(c)
except ZeroDivisionError as error:
    print(error)
finally:
    print('Finishing up.')
```

Code language: PHP (php)

Output:

```
5.0
Finishing up.
```

Source: https://www.pythontutorial.net/python-basics/python-try-except-finally/

Decorators

A decorator in Python is a function that modifies or adds extra features to another function without changing its code.

```
def decorator(func):
    def wrapper():
        print("Before function")
        func()
        print("After function")
    return wrapper
@decorator
def say_hello():
    print("Hello")
say_hello()
```

Output:

```
pgsql
Copy code
Before function
Hello
After function
```

OOPS

Python supports the core principles of object-oriented programming, which are the building blocks for designing robust and reusable software. The diagram below demonstrates these core principles:

1. Class: A class is a collection of objects. Classes are blueprints for creating objects. A class defines a set of attributes and methods that the created objects (instances) can have.

2. Objects: An Object is an instance of a Class. It represents a specific implementation of the class and holds its own data.

3. Inheritance: Inheritance allows a class (child class) to acquire properties and methods of another class (parent class). It supports hierarchical classification and promotes code reuse.

Types of Inheritance:

- **Single Inheritance:** A child class inherits from a single parent class.
- **Multiple Inheritance:** A child class inherits from more than one parent class.
- **Multilevel Inheritance:** A child class inherits from a parent class, which in turn inherits from another class.
- **Hierarchical Inheritance:** Multiple child classes inherit from a single parent class.
- **Hybrid Inheritance:** A combination of two or more types of inheritance.

4. Polymorphism: Polymorphism in Python means "same operation, different behavior." It allows functions or methods with the same name to work differently depending on the type of object they are acting upon.

5. Encapsulation: Encapsulation is the bundling of data (attributes) and methods (functions) within a class, restricting access to some components to control interactions. A class is an example of encapsulation as it encapsulates all the data that is member functions, variables, etc.

Types of Encapsulation:

- **Public Members:** Accessible from anywhere.
- **Protected Members:** Accessible within the class and its subclasses.
- **Private Members**: Accessible only within the class.

6. Data Abstraction: Abstraction hides the internal implementation details while exposing only the necessary functionality. It helps focus on "what to do" rather than "how to do it."

Types of Abstraction:

- **Partial Abstraction:** Abstract class contains both abstract and concrete methods.
- **Full Abstraction:** Abstract class contains only abstract methods (like interfaces).

List comprehension, dictionary comprehension

List comprehension in Python is a short and concise way to create lists. Instead of writing a loop and appending values, you can write everything in one line.

```
numbers = [x for x in range(5)]

print(numbers)
```

Output:

```
[0, 1, 2, 3, 4]
```

Python also has dictionary comprehension, Just like list comprehension, which lets you create dictionaries in a single line.

```
squares = {x: x**2 for x in range(5)}

print(squares)
```

Output:

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

Iterators & Generators

Iterators

An iterator is an object which contains a countable number of values and it is used to iterate over iterable objects like list, tuples, sets, etc. Iterators are implemented using a class and a local variable for iterating is not required here, It follows lazy evaluation where the evaluation of the expression will be on hold and stored in the memory until the item is called specifically which helps us to avoid repeated evaluation. As lazy evaluation is implemented, it requires only 1 memory location to process the value and when we are using a large dataset then, wastage of RAM space will be reduced the need to load the entire dataset at the same time will not be there.

Using an iterator-

- iter() function is used to create an iterator containing an iterable object.
- next() function is used to call the next element in the iterable object.
- After the iterable object is completed, to use them again reassign them to the same object.

```
iter_list = iter(['Geeks', 'For', 'Geeks'])

print(next(iter_list))

print(next(iter_list))

print(next(iter_list))
```

Output:

```
Geeks

For

Geeks
```

Generators

It is another way of creating iterators in a simple way where it uses the keyword “yield” instead of returning it in a defined function. Generators are implemented using a function. Just as iterators, generators also follow lazy evaluation. Here, the yield function returns the data without affecting or exiting the function. It will return a sequence of data in an iterable format where we need to iterate over the sequence to use the data as they won’t store the entire sequence in the memory.

```
def sq_numbers(n):
  for i in range(1, n+1):
    yield i*i
```

```
a = sq_numbers(3)
print("The square of numbers 1,2,3 are : ")
print(next(a))
print(next(a))
print(next(a))
```

Output:

```
The square of numbers 1,2,3 are :
1
4
9
```

Source: https://www.geeksforgeeks.org/python/difference-between-iterator-vs-generator/

- **Optional**

Virtual environments & pip

A virtual environment in Python is an isolated environment on your computer, where you can run and test your Python projects.It allows you to manage project-specific dependencies without interfering with other projects or the original Python installation.

Think of a virtual environment as a separate container for each Python project. Each container:

- Has its own Python interpreter
- Has its own set of installed packages
- Is isolated from other virtual environments
- Can have different versions of the same package

Using virtual environments is important because:

- It prevents package version conflicts between projects
- Makes projects more portable and reproducible
- Keeps your system Python installation clean
- Allows testing with different Python versions

PIP

Package refers to a distribution of Python code that includes one or more modules or libraries. These packages are typically published on the Python Package Index (PyPI) and can be easily installed using pip. Python packages may contain modules, sub-packages, and additional resources such as documentation and data files.

Python PIP is the package manager for Python packages. We can use PIP to install packages that do not come with Python. The basic syntax of PIP commands in the command prompt is:

pip 'arguments'

Source: https://www.geeksforgeeks.org/python/python-pip/

Standard libraries

The Python Standard Library contains the exact syntax, semantics, and tokens of Python. It contains built-in modules that provide access to basic system functionality like I/O and some other core modules. Most of the Python Libraries are written in the C programming language. The Python standard library consists of more than 200 core modules. Let's have a look at some of the commonly used libraries:

1. TensorFlow: This library was developed by Google in collaboration with the Brain Team. It is an open-source library used for high-level computations. It is also used in machine learning and deep learning algorithms. It contains a large number of tensor operations. Researchers also use this Python library to solve complex computations in Mathematics and Physics.
2. Matplotlib: This library is responsible for plotting numerical data. And that's why it is used in data analysis. It is also an open-source library and plots high-defined figures like pie charts, histograms, scatterplots, graphs, etc.
3. Pandas: Pandas are an important library for data scientists. It is an open-source machine learning library that provides flexible high-level data structures and a variety of analysis tools. It eases data analysis, data manipulation, and cleaning of data. Pandas support operations like Sorting, Re-indexing, Iteration, Concatenation, Conversion of data, Visualizations, Aggregations, etc.
4. Numpy: The name "Numpy" stands for "Numerical Python". It is the commonly used library. It is a popular machine learning library that supports large matrices and multi-dimensional data. It consists of in-built mathematical functions for easy computations. Even libraries like TensorFlow use Numpy internally to perform several operations on tensors. Array Interface is one of the key features of this library.

5. SciPy: The name "SciPy" stands for "Scientific Python". It is an open-source library used for high-level scientific computations. This library is built over an extension of Numpy. It works with Numpy to handle complex computations. While Numpy allows sorting and indexing of array data, the numerical data code is stored in SciPy. It is also widely used by application developers and engineers.
6. Scrapy: It is an open-source library that is used for extracting data from websites. It provides very fast web crawling and high-level screen scraping. It can also be used for data mining and automated testing of data.
7. Scikit-learn: It is a famous Python library to work with complex data. Scikit-learn is an open-source library that supports machine learning. It supports variously supervised and unsupervised algorithms like linear regression, classification, clustering, etc. This library works in association with Numpy and SciPy.
8. PyGame: This library provides an easy interface to the Standard Directmedia Library (SDL) platform-independent graphics, audio, and input libraries. It is used for developing video games using computer graphics and audio libraries along with Python programming language.
9. PyTorch: PyTorch is the largest machine learning library that optimizes tensor computations. It has rich APIs to perform tensor computations with strong GPU acceleration. It also helps to solve application issues related to neural networks.
10. PyBrain: The name "PyBrain" stands for Python Based Reinforcement Learning, Artificial Intelligence, and Neural Networks library. It is an open-source library built for beginners in the field of Machine Learning. It provides fast and easy-to-use algorithms for machine learning tasks. It is so flexible and easily understandable and that's why is really helpful for developers that are new in research fields.

There are many more libraries in Python. We can use a suitable library for our purposes. Hence, Python libraries play a very crucial role and are very helpful to the developers.

Source: https://www.geeksforgeeks.org/python/libraries-in-python/

- **Coding Standards:**

Naming convention

Naming conventions in Python refer to rules and guidelines for naming variables, functions, classes, and other entities in your code. Adhering to these conventions ensures consistency, readability, and better collaboration among developers.

Python Naming Conventions

Here, we discuss the Naming Conventions in Python which are as follows.

- Modules
- Variables
- Classes
- Exceptions

Modules in Python are files containing Python definitions and statements. When naming a module, use lowercase letters and underscores, making it descriptive but concise.

Global variable should be in uppercase with underscores separating words, while locals variable should follow the same convention as functions. Demonstrating consistency in naming conventions enhances code readability and maintainability, contributing to a more robust and organized codebase.

Classes in Python names should follow the CapWords (or CamelCase) convention. This means that the first letter of each word in the class name should be capitalized, and there should be no underscores between words.This convention helps improve code readability and consistency in programming projects.

Exception in Python names should end with "Error," following the CapWords convention**.** it is advisable to choose meaningful names that reflect the nature of the exception, providing clarity to developers who may encounter the error**.**

Source: https://www.geeksforgeeks.org/python/python-naming-conventions/

Doctring

Docstrings (Documentation Strings) are special strings used to document Python code. They provide a description of what a module, class, function or method does.

- Declared using triple quotes (' ' ' or " " ").
- Written just below the definition of a function, class, or module.
- Unlike comments (#), docstrings can be accessed at runtime using __doc__ or help().

```
def greet(name):
    """This function greets the user by their name."""
    return f"Hello, {name}!"
```

Here, the docstring explains what the function does in one simple line.

Source: https://www.geeksforgeeks.org/python/python-docstrings/

Comments

Comments in Python are the lines in the code that are ignored by the interpreter during the execution of the program.

In Python, single line comments starts with hashtag symbol **#.**

```
# sample comment

name = "geeksforgeeks"

print(name)
```

Output

```
Geeksforgeeks
```

Source: https://www.geeksforgeeks.org/python/python-comments/

Types of testing

While writing code, everyone make mistakes and hence, Python testing is very important.

Python Testing Strategies

- Unit Testing: Explain concept of unit testing and its focus on testing individual components or units of code in isolation.
- Integration Testing: Discuss integration testing and its role in testing interactions between different components or modules within an application.
- Functional Testing: Explore functional testing and its emphasis on testing the functionality and behavior of an application from an end-user perspective.
- Acceptance Testing: Introduce acceptance testing and its focus on verifying that the application meets the specified requirements and user expectations.
- Exploratory Testing: Touch on exploratory testing as an ad-hoc and unscripted approach to testing that emphasizes human intuition and creativity.

Unit Testing Frameworks

1. Unittest: Unittest is Python's built-in testing framework inspired by JUnit. Tests are written in classes that inherit from unittest.TestCase, with assertions like assertEqual or assertTrue to check outcomes. It supports automatic test discovery and works well with other frameworks like pytest and doctest.

2. Pytest: Pytest is a popular third-party framework known for its simple, function-based tests and powerful features. It uses plain assert statements, supports fixtures, parameterized testing and has rich plugins for coverage, reporting and test isolation. It automatically discovers tests and can run unittest and doctest tests too.

3. Nose/Nose 2: Nose extends unittest to make testing easier, adding features like automatic test discovery and parallel execution. It provides many plugins for coverage, output capture and isolation. Though less popular today, it still offers compatibility with existing test suites written using unittest or doctest.

4. Doctest: Doctest allows you to write tests inside docstrings, making examples double as tests. It runs code snippets, compares the output with the expected result and ensures documentation stays accurate. It's lightweight, great for small functions and integrates easily with frameworks like unittest and pytest.

Source: https://www.geeksforgeeks.org/python/python-testing/

PEP8

PEP 8 is the official style guide for Python code. PEP stands for Python Enhancement Proposal, which is a document that suggests improvements or standards for Python. PEP 8 specifically defines the coding conventions (rules for writing Python code) so that code is:

- Readable
- Consistent
- Easy to maintain

SOLID and DRY principles

SOLID principle is a set of five design principles in object-oriented programming. They help make code cleaner, easier to maintain, and more scalable.

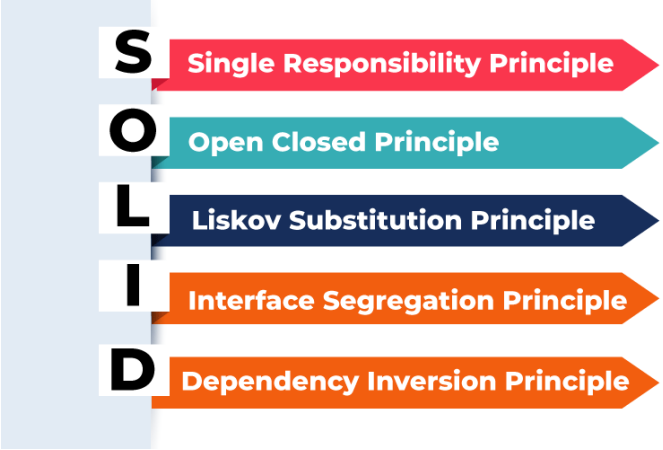
1. Single Responsibility Principle (SRP): A class should do only one job and have only one reason to change.

2. Open/Closed Principle (OCP): Code should be open for extension but closed for modification. That means you should be able to add new features without rewriting existing code.

3. Liskov Substitution Principle (LSP): Subclasses should be replaceable with their parent class without breaking the program.

4. Interface Segregation Principle (ISP): Don't force a class to implement methods it doesn't need.

5. Dependency Inversion Principle (DIP): High-level modules should not depend directly on low-level modules. Both should rely on abstractions.



Source: https://www.geeksforgeeks.org/system-design/solid-principle-in-programming-understand-with-real-life-examples/

DRY principle stands for Don't Repeat Yourself. Every piece of knowledge or logic in your code should exist only once. If you find yourself writing the same code multiple times, you should refactor it into a function, class, or module and reuse it.

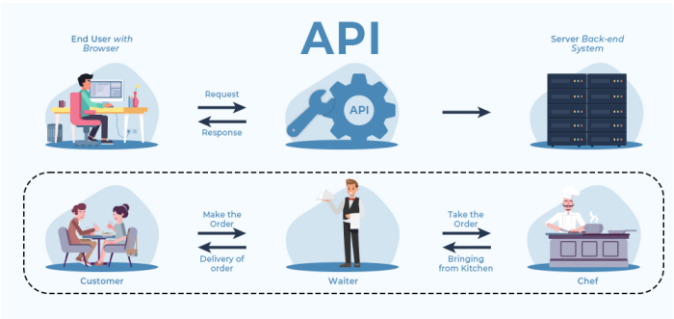
Why it matters

- Makes code easier to maintain: change in one place updates everywhere.
- Reduces bugs: no risk of forgetting to update a duplicated piece.
- Improves readability and reusability.

- **APIs Topics:**

API

An API is a set of rules that allow different software applications to communicate with each other . To understand it better, imagine you're at a restaurant: the waiter (API) takes your order (request), gives it to the chef (server), and then brings the prepared food (response) back to your table. Similarly, when you search for a course on a website, your request goes through an API, which then fetches the data from the database and sends it back as a response.



Types of APIs

There are three basic forms of API -

1. WEB APIs: A Web API also called Web Services is an extensively used API over the web and can be easily accessed using the HTTP protocols. A Web application programming interface is an open-source interface and can be used by a large number of clients through their phones, tablets, or PCs.

2. LOCAL APIs: In this type of API, the programmers get the local middleware services. TAPI (Telephony Application Programming Interface), and .NET are common examples of Local APIs.

3. PROGRAM APIs:mIt makes a remote program appear to be local by making use of RPCs (Remote Procedural Calls). SOAP is a well-known example of this type of API.

Few other types of APIs:

- SOAP (SIMPLE OBJECT ACCESS PROTOCOL): It defines messages in XML format used by web applications to communicate with each other.
- REST (Representational State Transfer): It makes use of HTTP to GET, POST, PUT, or DELETE data. It is basically used to take advantage of the existing data.
- JSON-RPC: It uses JSON for data transfer and is a lightweight remote procedural call defining a few data structure types.
- XML-RPC: It is based on XML and uses HTTP for data transfer. This API is widely used to exchange information between two or more networks.

Source: https://www.geeksforgeeks.org/software-testing/what-is-an-api/

HTTP Status codes

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

1. **Informational responses (100 – 199)**
2. **Successful responses (200 – 299)**
3. **Redirection messages (300 – 399)**
4. **Client error responses (400 – 499)**
5. **Server error responses (500 – 599)**

- 100 Continue
- 101 Switching Protocols

- 102 Processing
- 103 Early Hints
- 200 OK
- 201 Created
- 202 Accepted
- 203 Non-Authoritative Information
- 204 No Content
- 205 Reset Content
- 206 Partial Content
- 207 Multi-Status
- 208 Already Reported
- 226 IM Used
- 300 Multiple Choices
- 301 Moved Permanently
- 302 Found
- 303 See Other
- 304 Not Modified
- 307 Temporary Redirect
- 308 Permanent Redirect
- 400 Bad Request
- 401 Unauthorized
- 402 Payment Required
- 403 Forbidden
- 404 Not Found
- 405 Method Not Allowed
- 406 Not Acceptable
- 407 Proxy Authentication Required
- 408 Request Timeout
- 409 Conflict

- 410 Gone
- 411 Length Required
- 412 Precondition Failed
- 413 Content Too Large
- 414 URI Too Long
- 415 Unsupported Media Type
- 416 Range Not Satisfiable
- 417 Expectation Failed
- 418 I'm a teapot
- 421 Misdirected Request
- 422 Unprocessable Content
- 423 Locked
- 424 Failed Dependency
- 425 Too Early
- 426 Upgrade Required
- 428 Precondition Required
- 429 Too Many Requests
- 431 Request Header Fields Too Large
- 451 Unavailable For Legal Reasons
- 500 Internal Server Error
- 501 Not Implemented
- 502 Bad Gateway
- 503 Service Unavailable
- 504 Gateway Timeout
- 505 HTTP Version Not Supported
- 506 Variant Also Negotiates
- 507 Insufficient Storage
- 508 Loop Detected
- 510 Not Extended

- 511 Network Authentication Required

Source: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status

Response Formats

JSON (JavaScript Object Notation): JSON is a lightweight, human-readable data interchange format widely used in API responses due to its simplicity and flexibility. It represents data as key-value pairs, making it easy to parse and manipulate in various programming languages. JSON responses are well-suited for web APIs, mobile applications, and other scenarios requiring efficient data transfer. An example of JSON response looks like:

```
{
  "id": 123,
  "name": "John Doe",
  "email": "john@example.com",
  "age": 30
}
```

XML (eXtensible Markup Language): XML is another widely adopted format for representing structured data in API responses. Unlike JSON, XML utilizes tags to define hierarchical data structures, providing a more verbose but structured representation. While JSON is preferred for its simplicity and readability, XML remains relevant in certain domains, such as enterprise systems and legacy integrations.

```
<user>
  <id>123</id>
  <name>John Doe</name>
  <email>john@example.com</email>
  <age>30</age>
</user>
```

Other Formats (Optional): In addition to JSON and XML, APIs may utilize other response formats such as plain text, HTML, Protocol Buffers, or YAML, depending on specificrequirements and conventions within the domain. Each format has its own advantages and use cases, ranging from efficiency and performance to human readability and compatibility.

Source:https://apidog.com/blog/understanding-api-response-types-and-formats-a-comprehensive-guide/

Types of API Auth

Authentication: Refers to proving correct identity

Authorization: Refers to allowing a certain action

1. HTTP Authentication Schemes (Basic & Bearer)

This is the most straightforward method and the easiest. With this method, the sender places a username:password into the request header. The username and password are encoded with Base64, which is an encoding technique that converts the username and password into a set of 64 characters to ensure safe transmission.

2. API Keys

In REST API Security - API keys are widely used in the industry and became some sort of standard, however, this method should not be considered a good security measure.

3. OAuth (2.0)

The previous versions of this spec, OAuth 1.0 and 1.0a, were much more complicated than OAuth 2.0. The biggest change in the latest version is that it's no longer required to sign each call with a keyed hash.

4. OpenID Connect

OpenID Connect is a simple identity layer on top of the OAuth 2.0 protocol, which allows computing clients to verify the identity of an end-user based on the authentication performed by an authorization server, as well as to obtain basic profile information about the end-user in an interoperable and REST-like manner.

Versioning and Security

API Versioning is a defined process of making or managing changes to an API. These changes can be made transparently without interrupting the clients. When users have permission to decide whether they can conveniently upgrade to the latest version of API and clearly state the changes made is known as a Good API Versioning Strategy.When there is a condition or event of breaking change, then it is considered to be best practice to version your API because API versioning is a costly process for both the API users and API developers. A Breaking Change can be defined as a change that may be required to your software applications to avoid abnormal situations of application.

There are several types of strategies to versioning REST API using Spring Boot.

1. URI Path Versioning
2. Query Parameters Versioning

3. Custom Header Versioning
4. Content Negotiation Versioning

Source: https://www.geeksforgeeks.org/springboot/spring-boot-versioning-a-rest-api/

Security

Unsecured REST APIs expose sensitive data, allow unauthorized access, and create entry points for attacks like injection or data tampering.

Apply these techniques to protect your API from unauthorized access, data leaks, and misuse:

- **Use HTTPS** to encrypt all data in transit and prevent interception or tampering.
- **Enforce authentication** with API keys, tokens, or OAuth to block anonymous access.
- **Apply role-based access control** to restrict users to only the actions they're allowed to perform.
- **Validate and sanitize inputs** to stop injection attacks and prevent logic manipulation.
- **Limit request rates** with throttling or rate limiting to defend against brute-force and abuse.

Source: https://www.browserstack.com/guide/rest-api-design-principles-and-best-practices

CRUD operations

CRUD refers to the four basic operations a software application should be able to perform – Create, Read, Update, and Delete. In such apps, users must be able to create data, have access to the data in the UI by reading the data, update or edit the data, and delete the data. In full-fledged applications, CRUD apps consist of 3 parts: an API (or server), a database, and a user interface (UI). The API contains the code and methods, the database stores and helps the user retrieve the information, while the user interface helps users interact with the app.

In CRUD, the create operation does what the name implies. It means creating an entry. This entry could be an account, user information, a post, or a task.

The READ operation means getting access to the inputs or entries in the UI. That is, seeing it. Again, the entry could be anything from user information to social media posts, and others.

UPDATE is the operation that allows you to modify existing data. That is, editing the data.Unlike READ, the UPDATE operation alters the existing data by making changes to it.

To delete is to get rid of an entry from the UI and the database. DELETE is the HTTP protocol for implementing a DELETE operation.

Source: https://www.freecodecamp.org/news/crud-operations-explained/

Explore POSTMAN

Postman is an API (Application Programming Interface) development tool which helps to build, test and modify APIs. Almost any functionality that could be needed by any developer is encapsulated in this tool. It is used by over 5 million developers every month to make their API development easy and simple. It has the ability to make various types of HTTP requests(GET, POST, PUT, PATCH), saving environments for later use, converting the API to code for various languages (like JavaScript, Python).

Postman **started in 2014 as a side project of software engineer Abhinav Asthana**, a former intern at Yahoo Bangalore who found it difficult to develop and test APIs. He launched Postman as a free app in the Chrome Web Store.

Source: https://www.geeksforgeeks.org/software-testing/postman-tutorial/

Optimization and Efficiency

API Optimization

Making an API "optimized" means:

- Give only what's needed (not extra data).
- Break big data into smaller, easy-to-handle parts (pagination, filtering).
- Reuse results when possible (caching).
- Reduce unnecessary trips between client and server (batch requests).

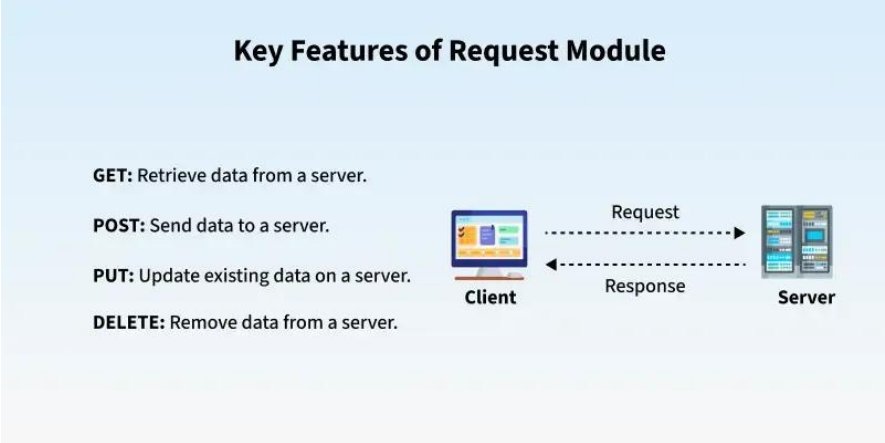
API Efficiency

Making an API "efficient" means:

- Handle many users at the same time without slowing down.
- Use the right HTTP methods properly (GET, POST, PUT, DELETE).
- Compress responses to save time and bandwidth.
- Keep backend/database organized so responses are quick.
- Offload heavy tasks so the system doesn't get stuck (asynchronous).

Requests lib in Python

Python Requests Library is a simple and powerful tool to send **HTTP** requests and interact with web resources. It allows you to easily send **GET**, **POST**, **PUT**, **DELETE**, **PATCH**, **HEAD** requests to web servers, handle responses, and work with REST APIs and web scraping tasks.



Source: https://www.geeksforgeeks.org/python/python-requests-tutorial/

RBAC

Role-Based Access Control (RBAC) is a security model that authorizes end-user access to systems, applications, and data based on predefined roles. For example, a security analyst can configure a firewall but cannot view customer data, while a sales representative can access customer accounts but not the firewall settings.

- Manages user access to protect resources—including data, applications, and systems—from unauthorized modification, addition, or deletion.
- Grants access based on a user's responsibilities according to their position.
- Defines roles and associated privileges, and assigns permissions to control access accordingly.

Core components of RBAC

1. Users – Individual accounts/persons.
2. Roles – Collections of permissions aligned to job functions.
3. Permissions – Allowed actions (e.g., read/write).
4. Role assignments – Users get one or more roles.

Source: https://www.geeksforgeeks.org/ethical-hacking/role-based-access-control-1/

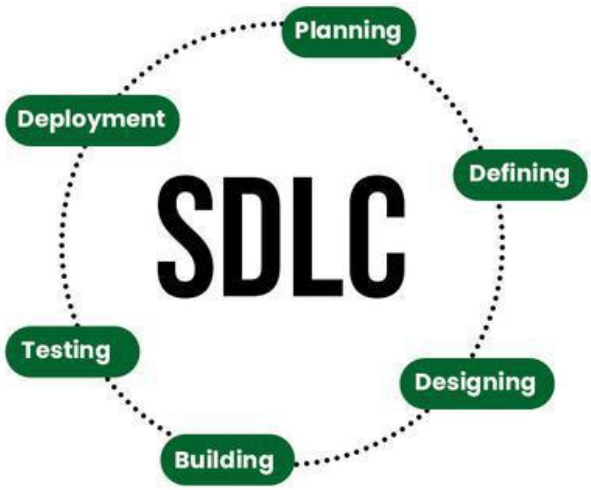
- **Extra Topics (Optional):**

SDLC

Software Development Life Cycle (SDLC) is a structured process that is used to design, develop, and test high-quality software. SDLC, or software development life cycle, is a

methodology that defines the entire procedure of software development step-by-step. The goal of the SDLC life cycle model is to deliver high-quality, maintainable software that meets the user's requirements.

SDLC in software engineering models outlines the plan for each stage so that each stage of the software development model can perform its task efficiently to deliver the software at a low cost within a given time frame that meets users requirements. In this article we will see Software Development Life Cycle (SDLC) in detail.



Source: https://www.geeksforgeeks.org/software-engineering/software-development-life-cycle-sdlc/

Agile Basics

The Agile methodology is a proper way of managing the project with breaking them into smaller phases which is iteration. It basically focus on flexibility of the project which we can change and improve the team work regularly as per requirements.

Agile is a Project Management and software development approach that aims to be more effective.

1. It focuses on delivering smaller pieces of work regularly instead of one big launch.
2. This allows teams to adapt to changes quickly and provide customer value faster.

Version Control

A Version Control System (VCS) is a tool used in software development and collaborative projects to track and manage changes to source code. It allows developers to:

- Record and track every update to the codebase
- Collaborate on code without overwriting each other's work

- Revert to earlier states of the project if needed
- Maintain a detailed and structured history of the project's evolution

If something breaks or doesn't work, the team can compare versions, isolate the issue, and restore stable code without starting from scratch.

Source: https://www.geeksforgeeks.org/git/version-control-systems/

Software Architecture

Software Architecture defines fundamental organization of a system and more simply defines a structured solution. It determines how the various components of a software system are assembled, how they relate to one another, and how they communicate. Essentially, it serves as a blueprint for the application and a foundation for the development team to build upon.

Software architecture defines a list of things which results in making many things easier in the software development process.

- **System structure**: The organization and arrangement of components.
- **System behavior**: The expected functionality and performance.
- **Component relationships**: How different parts of the system interact.
- **Communication structure**: The way components communicate with each other.
- **Stakeholder balance**: Meeting the needs and expectations of all stakeholders.
- **Team structure**: How the development team is organized and coordinated.
- **Early design decisions**: Making important choices early on to guide development.

Source: https://www.geeksforgeeks.org/software-engineering/fundamentals-of-software-architecture/