**Core Python:-**

**Loops and Clauses:-**

Loops and clauses are two important concepts in Python programming. Loops allow you to execute a block of code repeatedly, while clauses allow you to control the flow of execution within a loop.

There are two main types of loops in Python: for loops and while loops.

For loops are used to iterate over a collection of data.

While loops are used to execute a block of code repeatedly, as long as a certain condition is met.

There are a number of clauses that can be used to control the flow of execution within a loop. These clauses include:

* The break clause is used to terminate the loop immediately.
* The continue clause is used to skip the remainder of the current iteration of the loop and continue with the next iteration.
* The else clause is used to execute a block of code if the loop terminates normally.

**While-else construct:-**

The while-else construct in Python is a way to execute code after a while loop has finished. The else block will only be executed if the loop terminates normally, without being terminated by a break statement.

**For-else Construct:-**

The for-else construct in Python is a way to execute code after a for loop has finished. The else block will only be executed if the loop terminates normally, without being terminated by a break statement.

**Handing Search Failure with for-else:-**

def search(text, pattern):

for match in re.finditer(pattern, text):

yield match.start(), match.end()

else:

yield None, None

This function uses the re module to search for the given pattern in the given text. If the pattern is found, the function yields a tuple containing the start and end positions of the match. If the pattern is not found, the function yields None for both the start and end positions.

**Try Else Causes :-**

The else clause in a try statement in Python is executed if no exception occurs in the try clause. If an exception occurs, the else clause is not executed.

**Emulating Switch:-**

**Using a dictionary:**

One way to emulate a switch statement is to use a dictionary. The keys of the dictionary would be the different values that you want to check for, and the values of the dictionary would be the code that you want to execute if the value is matched.

# Create a dictionary of values and code

values = {

"foo": print("This is foo"),

"bar": print("This is bar"),

"baz": print("This is baz")

}

# Get the value from the user

value = input("Enter a value: ")

# Check the value and execute the corresponding code

if value in values:

values[value]()

else:

print("Invalid value")

**Using a function:-**

# Create a function that takes a value as an argument

def switch(value):

if value == "foo":

return print("This is foo")

elif value == "bar":

return print("This is bar")

elif value == "baz":

return print("This is baz")

else:

return print("Invalid value")

# Get the value from the user

value = input("Enter a value: ")

# Execute the code that is returned by the function

switch(value)()

**Using a class:**

Finally, you can also emulate a switch statement by using a class. The class would have a method for each value that you want to check for, and the method would execute the code that you want to execute if the value is matched.

# Create a class with a method for each value

class Switch:

def \_\_init\_\_(self, value):

self.value = value

def foo(self):

print("This is foo")

def bar(self):

print("This is bar")

def baz(self):

print("This is baz")

# Get the value from the user

value = input("Enter a value: ")

# Create an instance of the class and call the corresponding method

switch = Switch(value)

switch.method()

**Coalescing Nulls:-**

**Using the or operator:**

One way to coalescing nulls is to use the or operator. The or operator will return the first value that is not None.

# Get a value from the user

value = input("Enter a value: ")

# Coalesce the value with None

coalesced\_value = value or None

# Print the coalesced value

print(coalesced\_value)

**Using the coalesce() function:**

Another way to coalescing nulls is to use the coalesce() function. The coalesce() function takes two or more values as arguments, and it will return the first value that is not None.

# Get a value from the user

value = input("Enter a value: ")

# Coalesce the value with None using the coalesce() function

coalesced\_value = coalesce(value, None)

# Print the coalesced value

print(coalesced\_value)

**Using the default() function:**

Finally, you can also coalescing nulls by using the default() function. The default() function takes a value and a default value as arguments, and it will return the value if it is not None. If the value is None, then the default() function will return the default value.

# Get a value from the user

value = input("Enter a value: ")

# Coalesce the value with None using the default() function

coalesced\_value = default(value, None)

# Print the coalesced value

print(coalesced\_value)

**Abstract Base Class:-**

An abstract class is a class that cannot be instantiated directly. It can only be inherited from. Abstract classes are used to define a common interface for a set of classes. This interface can then be implemented by the child classes.

To define an abstract class in Python, you use the abc module. The abc module provides the ABCMeta metaclass, which is used to create abstract classes.

**Applications of ABCs:-**

* ABCs are a powerful tool that can be used to improve the quality of software.
* They can be used to define the common interface for a set of classes, to model the relationships between different entities in a data model, to create test cases, and to document the code.
* In addition to these specific examples, ABCs can be used in a variety of other ways to improve the quality of software.