Python Programming: Branching, Looping, and Functions

1. Branching (Conditional Statements)

Branching allows a program to execute different blocks of code based on conditions.

1.1 if Statement

Executes a block if the condition is True.

python

CopyEdit

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

1.2 if-else Statement

Executes different blocks for True or False conditions.

python

CopyEdit

```
x = 10
if x % 2 == 0:
    print("Even number")
else:
    print("Odd number")
```

1.3 if-elif-else Statement

Checks multiple conditions.

python

CopyEdit

```
x = 10
if x > 20:
    print("Greater than 20")
elif x > 5:
```

print("Greater than 5 but less than or equal to 20")

else:

print("5 or less")

1.4 Ternary Operator (Conditional Expression)

Compact if-else alternative.

python

CopyEdit

x = 10

y = 20

min_value = x if x < y else y # Returns the smaller value

print(min_value) # Output: 10

---

2. Looping (Iteration)

Loops are used to execute a block of code multiple times.

2.1 while Loop

Executes as long as the condition is True.

python

CopyEdit

i = 1

while i <= 5:

print("Iteration:", i)

i += 1 # Increment

2.2 for Loop

Iterates over a sequence (list, tuple, string, etc.).

python

CopyEdit

for i in range(1, 6):

```
    print("Iteration:", i)
```

Looping Over Different Data Types

python

CopyEdit

```
# Looping over a string
for char in "Python":
    print(char)


# Looping over a list
for item in [1, 2, 3, 4]:
    print(item)
```

2.3 Nested Loops

A loop inside another loop.

python

CopyEdit

```
for i in range(1, 4):
    for j in range(1, 4):
        print(i, j)
```

---

3. Exit Functions

Exit functions are used to terminate loops or functions.

3.1 break Statement

Exits the loop immediately.

python

CopyEdit

```
for i in range(1, 10):
    if i == 5:
```

```
        break
    print(i)  # Stops when i = 5
```

3.2 continue Statement

Skips the current iteration and continues with the next.

python

CopyEdit

```
for i in range(1, 6):
    if i == 3:
        continue  # Skips 3
    print(i)
```

3.3 pass Statement

A placeholder used when a statement is required but no code is executed.

python

CopyEdit

```
for i in range(5):
    if i == 2:
        pass  # Placeholder, does nothing
    print(i)
```

---

4. Functions in Python

A function is a reusable block of code that performs a specific task.

4.1 Defining a Function

python

CopyEdit

```
def greet():
    print("Hello, Welcome to Python!")
```

greet() # Function call

4.2 Function with Parameters

python

CopyEdit

```
def add(a, b):
    return a + b


result = add(5, 3)
print(result)  # Output: 8
```

4.3 Default Arguments

If no value is provided, the function uses a default argument.

python

CopyEdit

```
def greet(name="Guest"):
    print(f"Hello, {name}!")


greet()      # Output: Hello, Guest!
greet("John") # Output: Hello, John!
```

4.4 Keyword Arguments

Pass arguments using parameter names.

python

CopyEdit

```
def info(name, age):
    print(f"Name: {name}, Age: {age}")


info(age=25, name="Alice")  # Order doesn't matter
```

4.5 Variable-Length Arguments

Used when the number of arguments is unknown.

Arbitrary Arguments (args)

Used for multiple positional arguments.

python

CopyEdit

```
def add_all(*numbers):
    return sum(numbers)


print(add_all(1, 2, 3, 4, 5))  # Output: 15
```

****Arbitrary Keyword Arguments (kwargs)**

Used for multiple keyword arguments.

python

CopyEdit

```
def display_info(**details):
    for key, value in details.items():
        print(f"{key}: {value}")


display_info(name="Alice", age=25, city="New York")
```

---

5. Scope of Functions

Scope defines the visibility of variables inside and outside functions.

5.1 Local Scope

Variables declared inside a function are local to that function.

python

CopyEdit

```
def func():
    x = 10  # Local variable
```

```
    print(x)


func()
# print(x)  # Error! x is not accessible outside
```

5.2 Global Scope

Variables declared outside a function are global.

python

CopyEdit

```
x = 10  # Global variable


def func():
    print(x)  # Accessible inside function


func()
```

5.3 Global Keyword

To modify a global variable inside a function, use global.

python

CopyEdit

```
x = 10


def modify():
    global x
    x = 20  # Modifies global x


modify()
print(x)  # Output: 20
```

---

6. Function Documentation

Python provides docstrings to describe functions.

Docstring Example

python

CopyEdit

```
def square(n):
    """Returns the square of a number."""
    return n * n


print(square.__doc__)  # Output: Returns the square of a number.
```

---

7. Lambda Functions

A lambda function is an anonymous function defined using lambda.

Syntax

python

CopyEdit

```
lambda arguments: expression
```

Example

python

CopyEdit

```
square = lambda x: x * x
print(square(5))  # Output: 25
```

Lambda with Multiple Arguments

python

CopyEdit

```
add = lambda a, b: a + b
print(add(3, 7))  # Output: 10
```

---

8. Map Function

map() applies a function to all items in an iterable.

Syntax

python

CopyEdit

map(function, iterable)

Example

python

CopyEdit

```
numbers = [1, 2, 3, 4]
squared = list(map(lambda x: x**2, numbers))
print(squared)  # Output: [1, 4, 9, 16]
```

Conclusion

- **Branching (if-else) allows decision-making.**
- **Loops (for, while) execute repetitive tasks.**
- **Exit statements (break, continue, pass) control loops.**
- **Functions help in code reusability.**
- **Scope defines variable visibility.**
- **Lambda functions simplify function definitions.**
- **Map applies functions to iterables.**