Python Control Statements, Functions, and Recursion

1. Conditional Statements

Definition: Conditional statements are programming constructs that allow the execution of different code blocks based on whether a specified condition evaluates to True or False.

The if Statement

Definition: The if statement executes a block of code only when a specified condition evaluates to True.

Example:

python

```
age = 18

if age >= 18:
    print("You are eligible to vote")
```

The if-else Statement

Definition: The if-else statement provides two alternative paths of execution - one for when the condition is True, another for when it's False.

Example:

python

```
number = 7

if number % 2 == 0:
    print(f"{number} is even")
else:
    print(f"{number} is odd")
```

The if-elif-else Statement

Definition: The if-elif-else statement allows checking multiple conditions in sequence. Python evaluates each condition from top to bottom and executes the first block whose condition is True.

Example:

python

```
marks = 85

if marks >= 90:
    grade = "A"
elif marks >= 80:
    grade = "B"
elif marks >= 70:
    grade = "C"
else:
    grade = "F"
print(f"Your grade is: {grade}")
```

Nested if Statements

Definition: Nested if statements are conditional statements placed inside other conditional statements, creating a hierarchy of conditions.

Example:

python

```
age = 25
is_citizen = True

if age >= 18:
    if is_citizen:
        print("You can vote")
    else:
        print("You must be a citizen to vote")
else:
    print("You must be 18 or older")
```

2. Loops

Definition: Loops are control structures that allow repetitive execution of a block of code multiple times.

The for Loop

Definition: The for loop iterates over a sequence (list, tuple, string, range) or any iterable object, executing a block of code once for each item.

Example:

python

```
# Multiplication table

num = 5

for i in range(1, 11):
    print(f"{num} x {i} = {num * i}")
```

The while Loop

Definition: The while loop repeatedly executes a block of code as long as a specified condition remains True.

Example:

python

```
# Password validation

attempts = 3

password = "python123"


while attempts > 0:
    user_input = input("Enter password: ")
    if user_input == password:
        print("Login successful!")
        break
    else:
```

```
    attempts -= 1
    print(f"Wrong password. {attempts} attempts remaining")
```

break Statement

Definition: The break statement immediately exits a loop, regardless of the loop's condition.

Example:

python

```
# Exit when value found
numbers = [10, 20, 30, 40, 50]
search = 30
for num in numbers:
  if num == search:
    print(f"Found {search}")
    break
```

continue Statement

Definition: The continue statement skips the remaining code in the current iteration and immediately moves to the next iteration.

Example:

python

```
# Skip even numbers, print only odd
for i in range(1, 11):
  if i % 2 == 0:
    continue
  print(i)
# Output: 1, 3, 5, 7, 9
```

pass Statement

Definition: The pass statement is a null operation that does nothing. It acts as a placeholder where code is syntactically required but no action is needed.

Example:

python

```
for i in range(5):
    if i == 3:
        pass  # TODO: implement this later
    else:
        print(i)
```

---

3. Functions

Definition: A function is a reusable, self-contained block of code that performs a specific task. Functions take input (parameters), process it, and optionally return output.

Defining Functions

Definition: Defining a function means creating a named block of code using the def keyword that can be called later.

Example:

python

```
def calculate_area(radius):
    """Calculate the area of a circle"""
    pi = 3.14159
    return pi * radius ** 2


area = calculate_area(5)
print(f"Area: {area}")
```

Function Arguments

Definition: Function arguments are values passed to a function when it's called, allowing functions to work with different data.

1. Positional Arguments

Definition: Positional arguments must be passed in the exact order they are defined in the function.

Example:

python

```
def introduce(name, age, city):
    print(f"I am {name}, {age} years old, from {city}")


introduce("Alice", 25, "New York")
```

2. Keyword Arguments

Definition: Keyword arguments are passed with explicit parameter names, allowing arguments to be passed in any order.

Example:

python

```
def introduce(name, age, city):
    print(f"I am {name}, {age} years old, from {city}")


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

3. Default Arguments

Definition: Default arguments have predefined values in the function definition. If no value is provided, the default is used.

Example:

python

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


greet("Alice")  # Uses default: Hello, Alice!
greet("Bob", "Hi")  # Custom message: Hi, Bob!
```

4. Variable-length Arguments (args)

Definition: *args allows a function to accept any number of positional arguments, collecting them into a tuple.

Example:

python

```
def sum_all(*numbers):
    total = 0
    for num in numbers:
        total += num
    return total


print(sum_all(1, 2, 3))  # 6
print(sum_all(10, 20, 30, 40))  # 100
```

5. Keyword Variable-length Arguments (kwargs)**

Definition: **kwargs allows a function to accept any number of keyword arguments, collecting them into a dictionary.

Example:

python

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


print_details(name="Alice", age=25, city="NYC")
```

Scope of Variables

Definition: Scope refers to the region of a program where a variable is accessible.

1. Local Variables

Definition: Local variables are defined inside a function and exist only within that function.

Example:

python

```
def my_function():
    x = 10  # Local variable
    print(f"Inside: {x}")


my_function()
# print(x)  # Error: x not accessible outside
```

2. Global Variables

Definition: Global variables are defined outside any function and are accessible throughout the program.

Example:

python

```
x = 10  # Global variable


def my_function():
    print(f"Inside: {x}")


my_function()
print(f"Outside: {x}")
```

3. Using global keyword

Definition: The global keyword allows a function to modify a global variable.

Example:

python

```
count = 0
```

```
def increment():
    global count
    count += 1
    print(f"Count: {count}")

increment()  # Count: 1
increment()  # Count: 2
```

4. Nonlocal Variables

Definition: The nonlocal keyword allows an inner function to modify a variable from its enclosing function.

Example:

python

```
def outer():
    x = 10
    def inner():
        nonlocal x
        x = 20
    inner()
    print(f"x = {x}")  # x = 20

outer()
```

---

4. Lambda Functions

Definition: A lambda function is a small, anonymous function defined using the lambda keyword. It can have multiple parameters but only one expression.

Basic Lambda

Example:

python

```
square = lambda x: x ** 2
print(square(5))  # 25


add = lambda x, y: x + y
print(add(10, 20))  # 30
```