Loops and Iterations in Python

Key Concepts

- Loops
- Iterations
- `for` loop
- `while` loop
- Loop control statements (`break`, `continue`, `pass`)
- Nested loops

Introduction

In programming, loops are used to execute a block of code repeatedly until a certain condition is met. Loops are an essential part of programming as they allow us to perform repetitive tasks efficiently. Python provides two main types of loops: `for` loops and `while` loops.

Loops in Python

`for` Loop

The `for` loop is used to iterate over a sequence (such as a list, tuple, string, or range) and execute a block of code for each item in the sequence. The syntax of the `for` loop is as follows:

```
for item in sequence:
    statement1
    statement2
    ...
    statementN
```

Example:

```
fruits = ["apple", "banana", "cherry"] # List of fruits
for fruit in fruits:
    print(fruit)

# Output:
# apple
# banana
# cherry
```

In the above example, the `for` loop iterates over the list of fruits and prints each fruit on a new line.

`while` Loop

The `while` loop is used to execute a block of code as **long as a certain condition is `True`.** The syntax of the `while` loop is as follows:

```
while condition:
    statement1
    statement2
    ...
    statementN
```

Example:

```
count = 0
while count < 5:
    print(count)
    count += 1

# Output:
# 0
# 1
# 2
# 3
# 4
```

In the above example, the `while` loop prints the value of `count` as long as `count` is less than `5`.

Finite and Infinite Loops

- **Finite Loop**: A loop that executes a fixed number of times is called a finite loop. The number of iterations is known in advance. Example: `for` loop.
- **Infinite Loop**: A loop that continues to execute indefinitely is called an infinite loop. The loop condition never becomes `False`. Example: `while` loop without a proper termination condition.

Loop Control Statements

`break` Statement

The `break` statement is used to exit a loop prematurely. When the `break` statement is encountered inside a loop, the loop is terminated immediately, and the program control moves to the next statement after the loop. The `break` statement is often used to exit a loop when a certain condition is met.

Example:

```
for i in range(10): # Loop from 0 to 9, range(10) generates a list
ofnumbers from 0 to 9
    if i == 5:
        break
    print(i) # Print the value of i if i is not equal to 5

# Output:
# 0
# 1
# 2
# 3
# 4
```

In the above example, the loop is terminated when the value of `i` is equal to `5`.

`continue` Statement

The `continue` statement is used to skip the rest of the code inside a loop for the current iteration and move to the next iteration. When the `continue` statement is encountered inside a loop, the remaining statements inside the loop are skipped, and the loop continues with the next iteration.

Example:

```
# List of 10 random numbers between 1 and 75, this will represent
ages
ages = [23, 45, 12, 67, 34, 56, 18, 29, 42, 60]
for age in ages:
    if age < 18:
        continue # Skip the rest of the code for this iteration and
move to the next iteration
    print(f"Person aged {age} is an adult")

# Output:
# Person aged 23 is an adult
# Person aged 45 is an adult
# Person aged 67 is an adult
# Person aged 34 is an adult
# Person aged 56 is an adult
# Person aged 29 is an adult
# Person aged 42 is an adult
# Person aged 60 is an adult
```

In the above example, the loop skips the `print` statement for ages less than `18`.

Exercise: Write a program that prints all the even numbers between 1 and 20 using a `for` loop and the `continue` statement.