ST. JOSEPH'S
COLLEGE OF ENGINEERING
AND TECHNOLOGY,
- PALAI -
AUTONOMOUS

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ALGORITHMIC THINKING WITH PYTHON

Prof. Sarju S

17 October 2024

Module 3

Module 3

- SELECTION AND ITERATION USING PYTHON:- if-else, elif, for loop, range, while loop.
- SEQUENCE DATA TYPES IN PYTHON - list, tuple, set, strings, dictionary, Creating and using Arrays in Python (using Numpy library).
- DECOMPOSITION AND MODULARIZATION* :- Problem decomposition as a strategy for solving complex problems, Modularization, Motivation for modularization, Defining and using functions in Python, Functions with multiple return values

Prof. Sarju S, Department of Computer Science and Engineering, SJCET Palai

Whether you want to uncover the secrets of the universe, or you just want to pursue a career in the 21st century, basic computer programming is an essential skill to learn." – Stephen Hawking

SELECTION AND ITERATION USING PYTHON

if and if-else Statements

- The syntax of if statement in Python is:

```
if <condition>:
  # body of if statement
```

- Example

```
number = 10
# check if number is greater than 0
if number > 0:
    print('Number is positive.')
print('Statements outside if')
```

Following the logical condition, a **colon** : is required.

The body of the if **must be indented** and every line in this section of code must be indented the **same number of spaces**. **By convention, four space indentation is used in Python.** Most Python code editors automatically indent code after if-statements

if and if-else Statements

- An if statement can have an optional else clause.
- The syntax of if...else statement is:

```
if <condition>:
    # block of code if condition is True
else:
    # block of code if condition is False
```

if and if-else Statements

- Example

```
number = 10
if number > 0:
    print('Positive number')
else:
    print('Negative number')
print('This statement is always executed')
```

The keyword **else** should be on its own line and at the same indentation level as the **if** statement it belongs to. The **else** must be followed by a colon (:). Any code inside the **else** block should be indented by the same amount.

Multi-Way if Statements

Letter Grade	Range of Numeric Grades
A	All grades above 89
B	All grades above 79 and below 90
C	All grades above 69 and below 80
F	All grades below 70

Multi-Way if Statements

- The process of testing several conditions and responding accordingly can be described in code by a **multi-way selection** statement.

```
if <condition1>:
    # code block 1
elif <condition2>:
    # code block 2
else:
    # code block 3
```

In Python, the **elif** keyword is used to implement the "else if" statement. It allows you to check multiple conditions after an **if** statement, and it stands for "else if."

Multi-Way if Statements

- Example

```
number = 0
if number > 0:
    print("Positive number")
elif number == 0:
    print('Zero')
else:
    print('Negative number')
print('This statement is always executed')
```

Exercises

- Write a Python program that prompts the user to enter a number and checks if the number is even or odd. If the number is even, print "The number is even." If the number is odd, print "The number is odd."

👉 https://github.com/sarjus/Algorithemic-Thinking-with-Python-classroom-exercises/blob/main/evenodd.py

Exercises

- Write a Python program that takes two numbers as input from the user and prints the larger of the two numbers. If both numbers are equal, print "The numbers are equal."

👉 https://github.com/sarjus/Algorithemic-Thinking-with-Python-classroom-exercises/blob/main/largest_two_numbers.py

Exercises

- Write a Python program that takes a student's marks as input and prints their grade based on the following criteria:
  - Marks >= 90: A
  - Marks 80–89: B
  - Marks 70–79: C
  - Marks 60–69: D
  - Marks < 60: F

👉 https://github.com/sarjus/Algorithemic-Thinking-with-Python-classroom-exercises/blob/main/grade.py

Exercises

- Write a Python program that takes a single character as input from the user and checks if it is a vowel or a consonant. If the input is not an alphabetic character, print "Invalid input."

👉 https://github.com/sarjus/Algorithemic-Thinking-with-Python-classroom-exercises/blob/main/vowel_or_consonant.py

Exercises

- Write a Python program that takes a year as input and checks if the year is a leap year.
  - If the year is divisible by 400, it is a leap year.
  - If the year is divisible by 100 but not 400, it is not a leap year.
  - If the year is divisible by 4 but not 100, it is a leap year.

👉 https://github.com/sarjus/Algorithemic-Thinking-with-Python-classroom-exercises/blob/main/leap_year.py

Exercises

- Write a Python program that checks the strength of a password entered by the user. The program should categorize the password as:
  - "Weak" if it is less than 6 characters.
  - "Medium" if it is between 6 and 10 characters.
  - "Strong" if it is more than 10 characters.

👉 https://github.com/sarjus/Algorithemic-Thinking-with-Python-classroom-exercises/blob/main/password_strength.py

Repetition statements

- There are two types of loops—those that repeat an action a predefined number of times (**definite iteration**) and those that perform the action until the program determines that it needs to stop (**indefinite iteration**).

- Python's **for loop**, the control statement that most easily supports definite iteration.

```
for <variable> in range(<an integer expression>):
    <statement-1>
    .
    .
    <statement-n>
```

Loop body

Loop header

The colon (:) ends the loop header

Python range() Function

- The range() function returns a sequence of numbers, mainly used when working with for loops.
- The range() function can be represented in three different ways, or you can think of them as three range() parameters:
  - `range(stop_value)`: starting from 0 by default, and increments by 1 (by default) upto `stop_value`
  - `range(start_value, stop_value)`: This generates the sequence based on the start and stop value increments by 1 (by default).
  - `range(start_value, stop_value, step_size)`: It generates the sequence by incrementing the start value using the step size until it reaches the stop value.

Python range() Function

- Create a sequence of numbers from 0 to 5, and print each item in the sequence:

```
# Solution
for i in range(6):  # range(6) generates numbers from 0 to 5
    print(i)
```

- Create a sequence of numbers from 3 to 5, and print each item in the sequence:

```
# Solution
for i in range(3,6):  # range(3, 6) generates numbers from 3 to 5
    print(i)
```

Python range() Function

- Create a sequence of numbers from 3 to 19, but increment by 2 instead of 1:

```
# Solution
for i in range(3,20,2):  #range(start, stop, step)
    print(i)
```

Loops that Count Down

- When the **step argument** is a **negative** number, the range function generates a sequence of numbers from the first argument down to the second argument plus 1

```
>>> for count in range(10, 0, -1):
        print(count, end = " ")
10 9 8 7 6 5 4 3 2 1
```

Exercises

- Write the outputs of the following loops:

```
for count in range(5):
    print(count + 1, end = " ")
```

```
for count in range(1, 4):
    print(count, end = " ")
```

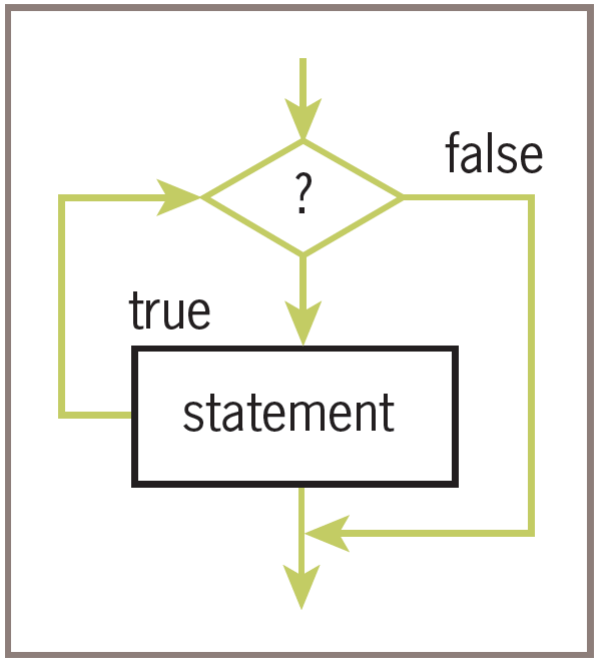
```
for count in range(1, 6, 2):
    print(count, end = " ")
```

```
for count in range(6, 1, -1):
    print(count, end = " ")
```

Conditional Iteration: The while Loop

- The Structure and behaviour of a while Loop

```
while <condition>:
      <sequence of statements>
```



Count Control with a while Loop

```
# Summation with a for loop
theSum = 0
for count in range(1, 100001):
      theSum += count
print(theSum)
```

```
# Summation with a while loop
theSum = 0
count = 1
while count <= 100000:
      theSum += count
      count += 1
print(theSum)
```

Loop control statements

- Python provides three loop control statements that control the flow of execution in a loop.
  - `break` statement
  - `continue` statement
  - `pass` statement

Loop control statements - break

- The **break** statement is used to immediately exit a loop, even if the loop condition is still true.
- When a **break** is encountered inside a loop, the control jumps out of the loop and the program continues with the next statement following the loop.

```
# Example: Find the first number greater than 5 and stop the loop
for num in range(1, 10):
    if num > 5:
        print(f"First number greater than 5 is: {num}")
        break  # Exit the loop when the first number greater than 5 is found
```

Output: First number greater than 5 is: 6

Loop control statements - continue

- The **`continue`** statement in Python is used to skip the rest of the code inside the current loop iteration and move on to the next iteration of the loop.
- Unlike **`break`**, which exits the loop completely, **`continue`** only skips the current iteration and allows the loop to proceed.

```
# Example: Skip printing even numbers and only print odd numbers
for num in range(1, 10):
    if num % 2 == 0:
        continue  # Skip the rest of the loop for even numbers
    print(num)
```

Output 

```
1
3
5
7
9
```

Loop control statements - pass

- The **`pass`** statement in Python is a placeholder used when a statement is required syntactically, but no action is needed.
- It allows you to write empty blocks of code without causing an error.
- The **`pass`** statement does nothing—it simply allows the program to continue running.

Loop control statements - pass

```
# Example: Function to check if a number is positive or negative and do nothing if it
is zero
num=int(input(“Enter a number”))
if num > 0:
        print(f"{num} is positive.")
elif num < 0:
        print(f"{num} is negative.")
else:
        pass  # Placeholder for when the number is zero; we will implement it later

print("This message will always print after the condition checks.")
```

Output

Enter a number 10
10 is positive.
This message will always print after the condition checks.

Enter a number0
This message will always print after the condition checks.

Difference between pass and continue

- **Functionality**
  - `pass`: Does nothing; serves as a placeholder.
  - `continue`: Skips to the next iteration of the loop.
- **Use Case**
  - `pass`: Useful when you need a block of code but haven't implemented it yet.
  - `continue`: Useful when you want to ignore certain conditions within a loop and proceed with the next iteration.
- **Behaviour**
  - `pass`: The program continues executing subsequent lines of code.
  - `continue`: The program jumps to the next iteration of the loop, skipping the remaining code in the current iteration.

Exercises

- Write a Python program to print numbers from 1 to 10 using a while loop.

👉 https://github.com/sarjus/Algorithemic-Thinking-with-Python-classroom-exercises/blob/main/loop_demo1.py

Exercises

- Write a Python program to calculate the sum of the first N natural numbers using a for loop. The user will input N.

👉 https://github.com/sarjus/Algorithemic-Thinking-with-Python-classroom-exercises/blob/main/sum_of_N_natural_numbers.py

Exercises

- Write a Python program that takes a number as input from the user and finds the factorial of that number using a while loop.

👉 https://github.com/sarjus/Algorithemic-Thinking-with-Python-classroom-exercises/blob/main/factorial_while_loop.py

Random Numbers

- Python's **random** module supports several ways to do this, but the easiest is to call the function **random.randint** with two integer arguments.

```
>>> import random
>>> for roll in range(10):
      print(random.randint(1, 6), end = " ")
2 4 6 4 3 2 3 6 2 2
```

Guessing game.

At start-up, the user enters the smallest number and the largest number in the range. The computer then selects a number from this range. On each pass through the loop, the user enters a number to attempt to guess the number selected by the computer. The program responds by saying “You’ve got it,” “Too large, try again,” or “Too small, try again.” When the user finally guesses the correct number, the program congratulates him and tells him the total number of guesses.

Guessing game.

```
import random
smaller = int(input("Enter the smaller number: "))
larger = int(input("Enter the larger number: "))
myNumber = random.randint(smaller, larger)
count = 0
while True:
        count += 1
        userNumber = int(input("Enter your guess: "))
        if userNumber < myNumber:
                print("Too small!")
        elif userNumber > myNumber:
                print("Too large!")
        else:
                print("Congratulations! You've got it in", count,"tries!")
                break
```

References

- https://www.freecodecamp.org/news/python-range-function-example/
- https://www.w3schools.com/python/ref_func_range.asp

ST.JOSEPH'S
COLLEGE OF ENGINEERING
AND TECHNOLOGY,
- PALAI -
AUTONOMOUS

Thank You

Prof. Sarju S
Department of Computer Science and Engineering
St. Joseph's College of Engineering and Technology, Palai (Autonomous)
sarju.s@sjcetpalai.ac.in