Conditional Statement, User Defined Functions & Iteration

Session Outline:

- IF & ELIF Statements.
- How to Define/Create a Function in Python?
- How to Create a Function with Parameters/Argument(s)? Function within a Function.
- Use Conditional Statements and Functions Together.
- Using FOR & WHILE Loops.
- Combining Conditional Statements & Loops.
- Combining Conditional Statements, Functions & Loops.
- How to Iterate over a Dictionaries

1. Conditional Statements: if, elif, else:

Conditional Statements: Used to make decisions in code based on conditions.

if condition1:

```
    # Code to execute if condition1 is True
```

elif condition2:

```
    # Code to execute if condition2 is True
```

else:

```
    # Code to execute if all conditions are False
```

In [ ]:
```
# Example: Conditional Statements
age = 18
if age < 13:
    print("You are a child.")
elif age < 20:
    print("You are a teenager.")
else:
    print("You are an adult.")
```

```
You are a teenager.
```

Real-life Use Case:

Age-based categorization or validating user input.

The **if statement** runs lines of indented code when a given logical condition is met

EXAMPLE *Determining experience level for a snowboard*

```
price = 999.99

if price > 500:
    print('This snowboard is designed for experienced users.')

print('This code will run whether or not the if statement is True.')
```

```
This product is for experienced users.
This code will run whether or not the if statement is True.
```

How does this code work?

CONTROL FLOW:

Control flow is a programming concept that allows you to choose which lines to execute, rather than simply running all the lines from top to bottom.

There are three conditional statements that help achieve this. **if**, **else**, and **elif**

Press tab, or use four spaces, to indent your code when writing if statements or you will receive an IndentationErro

The **else statement** runs lines of indented code when the none of the logical conditions in an if or elif statements are met

EXAMPLE *Determining experience level for a snowboard*

```
price = 499.99
price_threshold = 500

if price > price_threshold:
    print('This snowboard is for experienced users.')
else:
    print('This board is suitable for a beginner.')
```

```
This board is suitable for a beginner.
```

How does this code work?

The **elif statement** lets you specify additional criteria to evaluate when the logical condition in an if statement is not met

EXAMPLE *Determining experience level for a snowboard*

```
price = 499.99
expert_threshold = 500
intermediate_threshold = 300

if price > expert_threshold:
    print('This board is for experienced users.')
elif price > intermediate_threshold:
    print('This board is good for intermediate users.')
else:
    print('This should be suitable for a beginner.')
```

```
This board is good for intermediate users.
```

How does this code work?

NESTED IF STATEMENTS

Nested if statements let you specify additional criteria to evaluate after a logical condition in an if statement is met

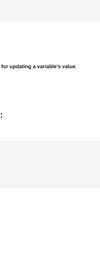
EXAMPLE *Trying to purchase a product*

```
price = 499.99
budget = 500
inventory = 0

if budget > price:
    print("You can afford this and we have is in stock!")
    if inventory == 0:  # equivalent to if inventory <= 0
        print("You can afford this but it's out of stock.")
else:
    print("Unfortunately, this board costs more than it(budget). ")

You can afford this but it's out of stock.
```

How does this code work?



2. User-Defined Functions:

Functions: Reusable blocks of code that perform a specific task.

Defining a Function: Use the def keyword.

Calling a Function: Use the function name followed by ().

In [ ]:
```
# Example: Defining and Calling a Function
def greet():
    print("Hello, World!")

greet()  # Calling the function
```

```
Hello, World!
```

Real-life Use Case:

Reusing code for repetitive tasks, such as calculating discounts.

3. Functions with Parameters/Arguments:

Parameters: Variables passed to a function.

Arguments: Values passed to the function when calling it.

In [ ]:
```
# Example: Function with Parameters
def greet_user(name):
    print(f"Hello, {name}!")

greet_user("Alice")  # Calling with an argument
```

```
Hello, Alice!
```

In [ ]:
```
def isPalindrome(s):
    return s == s[::-1]

s = "malayalam"
ans = isPalindrome(s)

if ans:
    print("Yes")
else:
    print("No")
```

Real-life Use Case:

Personalizing messages or processing user-specific data.

4. Function Within a Function:

Functions can call other functions or be nested.

In [ ]:
```
# Example: Function Within a Function

def outer_function():
    print("This is the outer function.")
    def inner_function():
        print("This is the inner function.")
    inner_function()

outer_function()
```

```
This is the outer function.
This is the inner function.
```

Real-life Use Case:

Modularizing complex tasks into smaller, reusable components.

5. Combining Conditional Statements and Functions:

Use conditions inside functions to make decisions.

In [ ]:
```
# Example: Combining Conditions and Functions
def check_even_odd(number):
    if number % 2 == 0:
        print(f"{number} is even.")
    else:
        print(f"{number} is odd.")

check_even_odd(10)
check_even_odd(7)
```

```
10 is even.
7 is odd.
```

Real-life Use Case:

Validating data or categorizing results.

6. Iteration: for and while Loops:

for Loop: Iterates over a sequence (e.g., list, string, range).

while Loop: Repeats as long as a condition is True.

In [ ]:
```
# Example: for Loop
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)

# Example: while Loop
count = 0
while count < 3:
    print("Count:", count)
    count += 1
```

```
apple
banana
cherry
Count: 0
Count: 1
Count: 2
```

In Python, the += operator is an **augmented assignment operator**, which is a **shorthand for updating a variable's value**.

Real-life Use Case:

Processing items in a list or repeating tasks until a condition is met.

7. Combining Conditional Statements and Loops:

Use conditions inside loops to control flow.

In [ ]:
```
# Example: Combining Conditions and Loops
numbers = [1, 2, 3, 4, 5]
for num in numbers:
    if num % 2 == 0:
        print(f"{num} is even.")
    else:
        print(f"{num} is odd.")
```

```
1 is odd.
2 is even.
3 is odd.
4 is even.
5 is odd.
```

Real-life Use Case:

Filtering or processing data based on conditions.

8. Combining Conditional Statements, Functions, and Loops:

Combine all three concepts for more complex tasks.

In [ ]:
```
# Example: Combining Conditions, Functions, and Loops

def print_squares(numbers):
    for num in numbers:
        print(f"Square of {num} is {num ** 2}.")

numbers = [1, 2, 3, 4, 5]
print_squares(numbers)
```

```
Square of 1 is 1.
Square of 2 is 4.
Square of 3 is 9.
Square of 4 is 16.
Square of 5 is 25.
```

Real-life Use Case:

Performing calculations or transformations on datasets.

9. Iterating Over Dictionaries:

Use .items(), .keys(), or .values() to iterate over dictionaries.

In [ ]:
```
# Example: Iterating Over Dictionaries

person = {
    "name": "Alice",
    "age": 25,
    "is_student": True
}

# Iterate over keys and values
for key, value in person.items():
    print(f"{key}: {value}")
```

```
name: Alice
age: 25
is_student: True
```

In [ ]:
```
# Iterate over keys
for key in person.keys():
    print("Key:", key)
```

```
Key: name
Key: age
Key: is_student
```

In [ ]:
```
# Iterate over values
for value in person.values():
    print("Value:", value)
```

```
Value: Alice
Value: 25
Value: True
```

Real-life Use Case:

Processing key-value pairs, such as user profiles or configurations.

10. Practice Exercise:

- Write a function that takes a list of numbers and returns the sum of even numbers.
- Use a while loop to repeatedly ask the user for input until they enter "quit".
- Create a dictionary of student grades and write a function to calculate the average grade.