

Assignment: Exploring Python Concepts

Hey there!

Welcome to your first Python assignment! 🎉 This is where the real fun begins. Today, we'll practice the fundamentals: variables, operators, and conditional statements. By the end of this assignment, you'll feel like a mini Python pro!

Here's what you need to do:

Task 1 - Variables: Your First Python Intro

Let's start simple! Imagine you're describing yourself (or anyone else you like) using Python variables.

1. Create a `name` variable that stores a string (like your name or a fictional character's name).
2. Create an `age` variable that stores an integer value.
3. Create a `height` variable that stores a floating-point number.

For example, something like this:

```
name = "Alex"  
age = 25  
height = 5.9
```

Now print these variables in a friendly message:

- **Example Output:** "Hey there, my name is Alex! I'm 25 years old and 5.9 feet tall."

Feel free to get creative with the message! 🚀

Task 2 - Operators: Playing with Numbers

We all love some math, don't we? Okay, maybe not everyone, but trust me, this will be easy and fun!

1. Pick two numbers, let's say `num1` and `num2` (you choose the values!).
2. Perform the following operations on these numbers:
 - Addition
 - Subtraction
 - Multiplication
 - Division

Write your Python code to calculate and display the results with a nice message for each.

For example:

```
num1 = 10
num2 = 3
print("The sum of 10 and 3 is", num1 + num2)
```

Be sure to explain what you're doing in comments! Bonus points if you throw in some humor. 😊

Task 3 - Conditional Statements: The Number Checker

Now for the real challenge: let's make your code think!

Write a program that takes a number from the user and tells them whether it's positive, negative, or zero.

Here's how it should work:

1. Ask the user to enter a number (use the `input()` function).
2. Use `if`, `elif`, and `else` statements to check:
 - If the number is greater than 0, print: "This number is positive. Awesome!"
 - If the number is less than 0, print: "This number is negative. Better luck next time!"
 - If the number is exactly 0, print: "Zero it is. A perfect balance!"

Project: Eligible Elector

Hey Python Wizard! 🧙

Let's make something cool and useful! You'll be writing a program to check voter eligibility. It's simple, but the result is super satisfying. Here's what you'll build:

Step 1: Ask the User's Age

Your program should start by asking the user:

```
age = int(input("How old are you? "))
```

Make sure you convert the input into an integer so you can use it in calculations later.

Step 2: Decide the Eligibility

Use a conditional statement to check if the age is 18 or older. Here's the logic:

- If the user's age is 18 or above, display a warm message like:
"Congratulations! You are eligible to vote. Go make a difference!"
 - If the user's age is less than 18, calculate how many years they have to wait. Display a friendly message like:
"Oops! You're not eligible yet. But hey, only X more years to go!"
-

Step 3: Test with Different Ages

Here's an example of how your program should behave:

Example Run 1:

```
How old are you? 16
```

```
Oops! You're not eligible yet. But hey, only 2 more  
years to go!
```

Example Run 2:

```
How old are you? 20
```

```
Congratulations! You are eligible to vote. Go make a  
difference!
```

Step 4: Polish It Up

Make sure your program is friendly, clear, and fun! Add comments to explain what your code is doing and maybe throw in a fun line or emoji for extra flair.

Extra Credit:

Want to stand out? Add some checks for invalid input (like negative ages or non-numeric entries). This is optional but will definitely impress!

Assignment: Explore Loops in Python

Hey there, Python enthusiast!

Get ready to dive deep into loops and unleash their full power! In this assignment, you'll practice using both `for` and `while` loops to solve practical problems. Let's get started!

Task 1 - Counting Down with Loops

Write a Python program to create a countdown timer.

1. Ask the user for a starting number.
2. Use a `while` loop to print numbers from that number down to 1.
3. When the countdown ends, print a celebratory message like "Blast off!"

For example:

```
Enter the starting number: 5  
5 4 3 2 1 Blast off! 🚀
```

Task 2 - Multiplication Table with `for` Loops

Write a program that generates the multiplication table for any number provided by the user.

1. Ask the user to input a number.
2. Use a `for` loop to print the multiplication table for that number (from 1 to 10).

Example Output:

```
Enter a number: 4
4 x 1 = 4 4 x 2 = 8 ... 4 x 10 = 40
```

Task 3 - Find the Factorial

Write a Python program to calculate the factorial of a number entered by the user.

1. Ask the user for a number.
2. Use a `for` loop to calculate the factorial.
3. Print the result in a friendly format.

For example:

```
Enter a number: 5
The factorial of 5 is 120.
```

Project: Number Guessing Game

Let's Make Loops Fun! 🎮

For this project, you'll create a simple yet addictive number-guessing game. Here's how it works:

Step 1: Generate a Random Number

Use Python's `random` module to generate a random number between 1 and 100.

```
import random
number_to_guess = random.randint(1, 100)
```

Step 2: Prompt the User for Guesses

Use a `while` loop to let the user keep guessing until they get the correct answer. After each guess:

- If the guess is too high, print: "Too high! Try again."

- If the guess is too low, print: "Too low! Try again."
 - If the guess is correct, print: "Congratulations! You guessed it!"
-

Step 3: Count the Attempts

Keep track of how many guesses the user has made and display the total number of attempts when they win.

Example Run:

```
Guess the number (between 1 and 100): 50 Too high! Try again.  
Guess the number (between 1 and 100): 25 Too low! Try again.  
Guess the number (between 1 and 100): 37 Congratulations! You  
guessed it in 3 attempts!
```

Bonus Task:

- Limit the number of attempts to 10. If the user fails to guess in 10 attempts, end the game with a message like "Game over! Better luck next time!"

Make your program friendly, interactive, and fun to play! 😊

Assignment: Exploring String Methods

Hey there, string wrangler!

Strings are one of the most exciting parts of Python, and this assignment is all about diving deep into them. Get ready to slice, dice, and manipulate strings like a pro!

Task 1 - String Slicing and Indexing

1. Create a string variable with the value "Python is amazing!".
2. Extract the following using slicing:
 - The first 6 characters ("Python")

- The word "amazing"
 - The entire string in reverse order
3. Print each of these slices with a clear label.

Example Output:

```
First word: Python
Amazing part: amazing
Reversed string: !gnizama si nohtyP
```

Task 2 - String Methods

1. Create a string with the value " hello, python world! ".
 2. Use the following string methods and print the results:
 - `strip()` to remove extra spaces
 - `capitalize()` to capitalize the first letter
 - `replace()` to replace "world" with "universe"
 - `upper()` to convert the string to uppercase
-

Task 3 - Check for Palindromes

Write a Python program to check if a string is a palindrome (reads the same backward and forward).

1. Ask the user to input a word.
2. Use slicing to reverse the string and compare it with the original.
3. Print a friendly message indicating whether the word is a palindrome.

Example Run:

```
Enter a word: madam
Yes, 'madam' is a palindrome!
```

Project: Password Strength Checker

Time to Get Practical!


In this project, you'll create a program to check the strength of a password. The goal is to make it informative and interactive.

Step 1: Input the Password

Ask the user to input a password. Use the `input()` function for this.

Step 2: Evaluate the Password

Check the password for the following:

1. Length: It should be at least 8 characters.
2. Contain at least one uppercase letter, one lowercase letter, one digit, and one special character (like `@`, `#`, `$`).
3. Print appropriate messages for each check:
 - If it passes all checks, print: "Your password is strong! "
 - If it fails any check, print a message like: "Your password needs at least one digit."

Use Python string methods like `isupper()`, `islower()`, `isdigit()`, and others to perform these checks.

Step 3: Test with Different Passwords

Here's how the program should behave:

Example Run 1:

```
Enter a password: python123
Your password needs at least one uppercase letter and
one special character.
```

Example Run 2:


```
Enter a password: Python@123
Your password is strong! 💪
```

Bonus Challenge:

Add a "password strength meter" that gives a score out of 10 based on how strong the password is.

Make sure your program is clear, friendly, and fun to use!

Assignment: Hands on Python Data Structures

Hey there, Python explorer!

Lists, dictionaries, and tuples are essential tools in Python, and this assignment will help you master their use. Let's get started!

Task 1 - Working with Lists

1. Create a list of your favorite fruits. Add at least five fruits to the list.
2. Perform the following operations:
 - Append a new fruit to the list.
 - Remove one fruit from the list using the `remove()` method.
 - Print the list in reverse order using slicing.

Example Output:

```
Original list: ['apple', 'banana', 'cherry', 'date', 'elderberry']
After adding a fruit: ['apple', 'banana', 'cherry', 'date', 'elderberry', 'fig']
After removing a fruit: ['banana', 'cherry', 'date', 'elderberry', 'fig']
```

```
Reversed list: ['fig', 'elderberry', 'date', 'cherry', 'banana']
```

Task 2 - Exploring Dictionaries

1. Create a dictionary to store information about yourself with the following keys: "name", "age", "city".
2. Add a new key-value pair to the dictionary for "favorite color".
3. Update the "city" key with a new value.
4. Print all the keys and values using a loop.

Example Output:

```
Keys: name, age, city, favorite color  
Values: Alice, 25, New York, Blue
```

Task 3 - Using Tuples

1. Create a tuple with three elements: your favorite movie, song, and book.
2. Try to change one of the elements (you'll see why tuples are immutable!).
3. Print the length of the tuple using the `len()` function.

Example Output:

```
Favorite things: ('Inception', 'Bohemian Rhapsody', '1984')  
Oops! Tuples cannot be changed.  
Length of tuple: 3
```

Project: Implement Your own Data Structures

Let's Get Real!

In this project, you'll create a simple inventory management program using lists, dictionaries, and tuples.

Step 1: Create the Inventory

1. Start with an empty dictionary called `inventory`.
2. Each key in the dictionary will represent an item name, and the value will be a tuple containing the quantity and price.

Example:

```
inventory =  
{  
    "apple": (10, 2.5),  
    "banana": (20, 1.2)  
}
```

Step 2: Add, Remove, and Update Items

1. Add functionality to:
 - Add a new item to the inventory (e.g., `"mango": (15, 3.0)`).
 - Remove an item from the inventory.
 - Update the quantity or price of an existing item.
-

Step 3: Display the Inventory

Write a loop to display all items in the inventory in a friendly format. For example:

```
Item: apple, Quantity: 10, Price: $2.5  
Item: banana, Quantity: 20, Price: $1.2
```

Step 4: Bonus - Calculate Total Value

Add a feature to calculate and display the total value of the inventory by multiplying the quantity and price of each item.

Example Run:

```
Welcome to the Inventory Manager!
Current inventory:
Item: apple, Quantity: 10, Price: $2.5
Item: banana, Quantity: 20, Price: $1.2
Adding a new item: mango
Updated inventory:
Item: apple, Quantity: 10, Price: $2.5
Item: banana, Quantity: 20, Price: $1.2
Item: mango, Quantity: 15, Price: $3.0
Total value of inventory: $90.0
```

Make sure your program is interactive, user-friendly, and easy to use. Add comments to explain your code for bonus clarity!

Assignment: About Parameters of Functions

Hello, Python Prodigy!

Functions and recursion are powerful tools that let us write reusable, efficient, and elegant code. Let's dive into these concepts with some engaging tasks.

Task 1 - Writing Functions

Create a function `greet_user` that accepts a name as a parameter and prints a personalized greeting.

Then, write another function `add_numbers` that takes two numbers as parameters, adds them, and returns the result.

Example Output:

plaintext

Copy code

```
Hello, Alice! Welcome aboard. The sum of 5 and 10 is 15.
```

Task 2 - Using Default Parameters

Create a function `describe_pet` that accepts two parameters:

- `pet_name` (string)
- `animal_type` (string, default value is "dog").

The function should print a description of the pet.

Example Output:

plaintext

Copy code

```
I have a dog named Buddy. I have a cat named Whiskers.
```

Task 3 - Functions with Variable Arguments

Write a function `make_sandwich` that accepts a variable number of arguments for sandwich ingredients and prints them as a list.

Example Output:

plaintext

Copy code

```
Making a sandwich with the following ingredients: -  
Lettuce - Tomato - Cheese
```

Task 4 - Understanding Recursion

Write a recursive function `factorial` to calculate the factorial of a number.

Then, write another recursive function `fibonacci` to calculate the nth number in the Fibonacci sequence.

Example Output:

plaintext

Copy code

```
Factorial of 5 is 120. The 6th Fibonacci number is 8.
```

Project: About Menu functioning

In this project, you'll create a Python program that showcases the beauty of recursion and the functionality of functions through a set of interactive and practical examples.

Step 1: Menu of Recursive Functions

Create a program that presents a menu of choices:

1. Calculate the factorial of a number.
2. Find the nth Fibonacci number.
3. Draw a recursive fractal pattern (bonus).
4. Exit.

The user should select an option by entering a number.

Step 2: Factorial Function

Implement a recursive function to calculate the factorial of a number and display the result.

Example Run:

plaintext

Copy code

```
Enter a number to find its factorial: 5 The
factorial of 5 is 120.
```

Step 3: Fibonacci Function

Implement a recursive function to calculate the nth Fibonacci number and display the result.

Example Run:

plaintext

Copy code

```
Enter the position of the Fibonacci number: 7
The 7th Fibonacci number is 13.
```

Step 4: Recursive Fractal Pattern (Bonus)

Using the `turtle` library, create a recursive function to draw a fractal pattern (e.g., a tree or snowflake).

Example Run: A simple fractal tree drawing! 

Step 5: User-Friendly Program

Enhance the program with the following features:

- Input validation (e.g., check for positive integers).
- Friendly prompts and error messages.
- Comments explaining the purpose of each function.

Example Run:

plaintext

Copy code

```
Welcome to the Recursive Artistry Program!  
Choose an option: 1. Calculate Factorial 2.  
Find Fibonacci 3. Draw a Recursive Fractal 4.  
Exit > 1 Enter a number to find its factorial:  
5 The factorial of 5 is 120.
```

Enjoy crafting your masterpiece with functions and recursion!

Assignment: Check your Knowledge on Errors

Hey there, Python Troubleshooter!

Exceptions and errors are essential topics for writing robust and fault-tolerant code. This assignment will guide you in handling errors gracefully and understanding Python exceptions. Let's begin!

Task 1 - Understanding Python Exceptions

Write a Python program that:

1. Prompts the user to enter a number.
2. Tries to divide 100 by the number.
3. Handles the following exceptions:
 - `ZeroDivisionError` (when dividing by zero).
 - `ValueError` (when the user enters non-numeric input).
4. Prints appropriate error messages for each exception.

Example Output:

```
Enter a number: 0 Oops! You cannot divide by zero.  
Enter a number: abc Invalid input! Please enter a  
valid number. Enter a number: 4 100 divided by 4 is  
25.0
```

Task 2 - Types of Exceptions

Create a program that intentionally raises and handles the following exceptions:

- `IndexError` by accessing an invalid list index.
- `KeyError` by trying to access a non-existent key in a dictionary.
- `TypeError` by adding a string and an integer.

Explain in comments how each error occurs and how it is handled.

Example Output:

```
IndexError occurred! List index out of range. KeyError
occurred! Key not found in the dictionary. TypeError
occurred! Unsupported operand types.
```

Task 3 - Using `try...except...else...finally`

Write a program that:

1. Prompts the user to enter two numbers.
2. Tries to divide the first number by the second number.
3. Implements the following:
 - `try` block to attempt the division.
 - `except` block to handle exceptions.
 - `else` block to display the result if no exceptions occur.
 - `finally` block to print a closing message regardless of exceptions.

Example Output:

```
Enter the first number: 10 Enter the second number: 2
The result is 5.0. This block always executes.
```

Project: Calculator with Exception Handling

In this project, you will create a Python calculator that gracefully handles exceptions and provides helpful feedback to users.

Step 1: Menu of Operations

Create a program that displays a menu with the following operations:

1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Exit

The user should select an option by entering a number.

Step 2: Input Validation

Add input validation to ensure the user enters numbers only for the calculations. If the input is invalid, catch the exception and prompt the user again.

Example Output:

plaintext

Copy code

```
Enter the first number: abc Invalid input! Please  
enter a valid number.
```

Step 3: Division with Exception Handling

Handle the following exceptions specifically for the division operation:

- `ZeroDivisionError`: Print a friendly message when dividing by zero.
- `ValueError`: Ensure the inputs are numbers.

Example Output:

plaintext

Copy code

```
Enter the first number: 10 Enter the second number: 0
Oops! Division by zero is not allowed.
```

Step 4: Logging Errors (Bonus)

Use the `logging` module to log errors to a file named `error_log.txt` whenever an exception occurs.

Example Log Entry:

plaintext

Copy code

```
ERROR:root:ZeroDivisionError occurred: division by
zero.
```

Step 5: User-Friendly Interface

Enhance the program with the following features:

- Clear prompts and instructions.
- A `try...except...else...finally` structure to ensure robustness.
- Informative error messages for each exception.

Example Run:

plaintext

Copy code

```
Welcome to the Error-Free Calculator! Choose an
operation: 1. Addition 2. Subtraction 3.
Multiplication 4. Division 5. Exit > 4 Enter the first
```

```
number: 10 Enter the second number: 0 Oops! Division
by zero is not allowed. Choose an operation: 5
Goodbye!
```

Overview: Capstone Project

Title: Python-Based Employee Management System Overview

Overview

The Python-Based Intelligent Study Planner is a capstone project focusing on creating a personalized, interactive application for learners to manage study tasks, track performance. It integrates Python programming concepts such as loops, data structures, exception handling.

Lessons

1. **Introduction to Python Fundamentals**
 - Basics of variables, loops, and conditional statements for task management.
1. **Error Handling**
 - Identifying and managing invalid inputs to enhance application stability.

Lesson: Problem Statements

Problem Statements as User Stories

User Story 1: Task Prioritization

As a student, I want to organize my tasks by deadlines and priority so that I can manage my study time efficiently.

- **Solution:** Allow input of tasks and automatically generate a prioritized schedule.

User Story 2: Performance Tracking

As a student, I want to track my scores across subjects so that I can identify areas for improvement.

- **Solution:** Enable score tracking and calculation of averages for actionable feedback.

Implementation Plan

1. Phase 1: Task Management

- Develop input methods for tasks and schedules.

1. Phase 2: Performance Tracking

- Implement score tracking and weak area identification.

1. Phase 3: Error Handling

- Manage invalid inputs and enhance application stability.

Learning Outcomes

1. Apply Python basics to solve real-world challenges.
2. Develop modular, error-resilient applications.