**1. Core Concepts & Syntax**

* **Variables:**
  + Containers for storing data. Assignment is done using the = operator. Variable names should be descriptive.
  + age = 25
  + student\_name = "Alice"
* **Data Types:**
  + int: Integers, e.g., 10, -5, 0.
  + float: Floating-point numbers (decimals), e.g., 9.81, 3.14.
  + str: Strings (text), which must be enclosed in single ' or double " quotes.
  + bool: Boolean values, which can only be True or False.
* **Input / Output:**
  + print(): Displays information in the console. It can print variables, text, or a combination.

Python

print("Hello, world!")

print("Your age is:", age)

input("Prompt message"): Gets input from the user. **Note: The return value of input() is always a string (str) type.**

Python

name = input("Please enter your name: ")

**Type Conversion:**

int(): Converts a value (usually a string) to an integer.

float(): Converts a value to a floating-point number.

str(): Converts a value to a string.

Python

age\_str = input("Enter your age: ") # age\_str is a string, e.g., "30"

age\_num = int(age\_str) # age\_num is an integer, e.g., 30

* **Basic Operators:**

**Arithmetic:** + (addition), - (subtraction), \* (multiplication), / (float division), // (integer division), % (modulo/remainder), \*\* (exponentiation).

**String Concatenation:** + is used to join two strings.

**2. Modules**

**Importing:** Use the import keyword to load a module's tools. All import statements should be placed at the top of the file.

Python

import turtle

**Accessing Module Content:** Use dot notation: module\_name.function\_name().

Python

window = turtle.Screen() # Access the Screen() function from the turtle module

**3. Turtle Graphics**

**Setup:**

window = turtle.Screen(): Creates the window (canvas) where the turtle will draw.

my\_turtle = turtle.Turtle(): Creates a turtle object, which is the "pen" you control.

* **Screen/Window Methods:**

window.bgcolor("color\_name"): Sets the background color of the window.

window.exitonclick(): Pauses the program and waits for a mouse click on the window to close it. This should be the last line of a turtle program.

* **Turtle Methods:**

**Movement:**

my\_turtle.forward(distance): Moves the turtle forward by a number of pixels.

my\_turtle.backward(distance): Moves the turtle backward.

my\_turtle.left(angle): Turns the turtle counter-clockwise by an angle in degrees.

my\_turtle.right(angle): Turns the turtle clockwise.

my\_turtle.goto(x, y): Moves the turtle to an absolute position on the screen. The center is (0, 0).

**Appearance & State:**

my\_turtle.color("color\_name"): Sets the turtle's pen and fill color.

my\_turtle.shape("shape\_name"): Changes the turtle's shape (e.g., "turtle", "arrow", "circle").

my\_turtle.penup(): Lifts the pen, so the turtle moves without drawing.

my\_turtle.pendown(): Puts the pen down to resume drawing.

**Drawing & Filling:**

my\_turtle.begin\_fill(): Call this before drawing a shape you want to be filled.

my\_turtle.end\_fill(): Call this after the shape is drawn to complete the fill.

**4. Control Flow**

**Conditional Statements (if/elif/else):**

Executes blocks of code based on whether a condition is true. elif (else if) and else are optional.

Python

score = 85

if score >= 90:

print("Grade A")

elif score >= 80:

print("Grade B")

else:

print("Grade C")

* **Boolean Logic:**

**Comparison Operators:** == (equal to), != (not equal to), <, >, <=, >=.

**Logical Operators:** and (both must be true), or (at least one must be true), not (inverts the boolean value).

* **for Loops:**

Iterates over a sequence (like a list or a range of numbers) a specific number of times.

Python

# Using range to repeat 4 times

for \_ in range(4):

my\_turtle.forward(100)

my\_turtle.right(90)

# Iterating through a list

colors = ["red", "blue", "green"]

for c in colors:

print(c)

* **range() Function:**

Generates a sequence of numbers, commonly used with for loops.

range(stop): From 0 up to (but not including) stop.

range(start, stop): From start up to stop.

range(start, stop, step): From start up to stop, incrementing by step.

* **while Loops:**

Repeats a block of code as long as a condition remains True. You must ensure the condition eventually becomes false to avoid an infinite loop.

Python

count = 0

while count < 5:

print(count)

count += 1 # Crucial: Update the loop variable

**5. Functions**

* **Defining a Function:**

A reusable block of code. Defined using the def keyword.

Can take inputs (parameters) and produce an output (return value).

Python

def greet(name):

"""A docstring explains what the function does."""

print("Hello, " + name)

* **Parameters & Arguments:**

**Parameter:** The variable listed inside the parentheses in the function definition (e.g., name).

**Argument:** The actual value sent to the function when it is called (e.g., "World" in greet("World")).

* **Returning a Value:**

The return statement exits a function and sends back a value. If there is no return statement, the function returns None.

Python

def add(a, b):

return a + b

sum\_result = add(5, 3) # sum\_result is now 8

* **main() Function Pattern:**

A common practice to organize your main program logic inside a function, typically called main().

Python

def main():

# All main program logic goes here

greet("class")

print("Program finished.")

# Call the main function to start the program

main()

**6. Data Structures: Lists**

**Creating a List:**

An ordered, mutable (changeable) collection of items. Created with square brackets [].

my\_list = [1, "hello", 3.14, True]

* **Accessing Elements (Indexing):**

Access items by their position, starting from 0. Negative indices access from the end.

fruits = ["apple", "banana", "cherry"]

print(fruits[0]) -> apple

print(fruits[-1]) -> cherry

* **Slicing:**

Extracts a portion of a list. list[start:stop]. The stop index is not included.

print(fruits[0:2]) -> ['apple', 'banana']

* **Common Functions and Methods:**

len(list): Returns the number of items in the list.

list.append(item): Adds an item to the very end of the list.

list.pop(): Removes and returns the last item.

list.pop(index): Removes the item at the specified index.

list.sort(): Sorts the list in place.

**7. Debugging**

* **Syntax Errors:**

The code violates the rules of the Python language (e.g., a missing colon :, incorrect indentation, mismatched parentheses). Python will refuse to run the code and will usually point to the location of the error.

* **Runtime Errors:**

Errors that occur while the program is running. The syntax is correct, but the program encounters an impossible situation (e.g., dividing by zero (ZeroDivisionError), trying to access a list item that doesn't exist (IndexError)). The program crashes and prints a "traceback".

* **Semantic Errors (Logical Errors):**

The code runs without crashing but does not produce the correct result. This is a flaw in the programmer's logic. These are often the hardest errors to find and require you to trace your code's execution step-by-step (print() statements can be very helpful here).