**Python Guide: Control Flow**

Welcome to your study guide on Control Flow in Python! This session covers the essential tools that allow your programs to make decisions and repeat actions. These concepts are the foundation of writing dynamic and intelligent code.

We will cover two main topics:

1. **Conditional Statements:** Using if, elif, and else to execute different blocks of code based on whether a condition is true or false.
2. **Loops:** Using the for loop to repeat a set of instructions multiple times.

**1. Conditional Statements (Making Decisions)**

Conditional statements allow a program to follow different paths based on certain conditions. Think of it as a fork in the road; the program checks a sign (the condition) and decides which way to go.

**The if-else Statement**

The if-else statement is the most basic decision-making structure. If a condition is true, the if block is executed. Otherwise, the else block is executed.

**Code Example: Leap Year Checker**

This program determines if a year entered by the user is a leap year.

# Write a program to checke given year is leap year or not

a = int(input("Enter the year: "))

if a%4 == 0 and a %100 != 0 or a%400 == 0:

print(a, "is a leap year")

else:

print(a, "is not a leap year")

**Explanation:**

1. a = int(input("Enter the year: ")): This line prompts the user to enter a year, reads the input as a string, and converts it into an integer.
2. The if statement checks a complex logical condition for leap years:
   * A year is a leap year if it is divisible by 4 (a%4 == 0)...
   * ...**AND** it is not a century year (a %100 != 0)...
   * ...**OR** it is a special century year divisible by 400 (a%400 == 0).
3. If this entire condition evaluates to True, the first print statement runs.
4. If the condition is False, the else block is executed.

**The if-elif-else Ladder**

When you have more than two possible outcomes, you can use the if-elif-else ladder to check a series of conditions in order. elif is short for "else if."

**Code Example: Grading System (Corrected Logic)**

Your original notes for this program contained some impossible conditions (e.g., a>=80 and a<70). Below is a corrected and more efficient version that achieves the goal.

Python

# Original Code from Py Day 8 had logical flaws. This is the corrected version.

a = int(input("Enter the percentage: "))

if a > 100 or a < 0:

print("Invalid Marks")

elif a >= 80:

print("A+")

elif a >= 70:

print("A")

elif a >= 60:

print("B")

elif a >= 50:

print("C")

elif a >= 35:

print("D")

else: # If none of the above, marks must be < 35

print("Fail")

**Explanation of the Corrected Logic:**

* The code checks conditions from top to bottom.
* Once a condition is met (e.g., if a is 75, a >= 70 is true), its block is executed, and the rest of the ladder is skipped. This makes it efficient.
* We only need to check one boundary (e.g., a >= 70) because if the code reaches that point, we already know that a is *not* greater than or equal to 80.

**Nested if-else Statements**

You can place if-else statements inside other if-else blocks. This is called nesting and is useful for checking a series of dependent conditions.

**Code Example: Nested Grading System**

This program from your notes achieves the same grading goal as the elif ladder but uses a nested structure.

Python

p = int(input("Enter Percentage: "))

if p <= 100 and p >= 0: # Outer check for valid percentage range

if p >= 80:

print("A+")

else:

if p >= 70:

print("A")

else:

if p >= 60:

print("B")

else:

if p >= 50:

print("C")

else:

if p >= 35:

print("D")

else:

print("Fail")

else:

print("Invalid Percentage")

**Explanation:**

* The outermost if statement first checks if the percentage is valid (between 0 and 100).
* If it is valid, the program enters a series of nested checks. Each else block contains another if statement to check the next lower grade boundary.
* This structure is functional but can become hard to read with deep nesting. The if-elif-else ladder is often preferred for its clarity.

**Code Example: Salary Bonus Calculation**

This nested if-else structure calculates a bonus based on different salary tiers.

Python

a = float(input("Enter Salary: "))

if a > 0:

if a >= 55000:

# 25% bonus

print(f"salary amt = {a} \\n bonus value = {a/100\*25} \\n total salary = {a+(a/100\*25)}")

else:

if a > 40000: # No need for a <= 55000, it's implied

# 15% bonus

print(f"salary amt = {a} \\n bonus value = {a/100\*15} \\n total salary = {a+(a/100\*15)}")

else:

if a > 30000:

# 10% bonus

print(f"salary amt = {a} \\n bonus value = {a/100\*10} \\n total salary = {a+(a/100\*10)}")

else:

if a > 20000:

# 7% bonus

print(f"salary amt = {a} \\n bonus value = {a/100\*7} \\n total salary = {a+(a/100\*7)}")

else:

if a > 8000:

# 5% bonus

print(f"salary amt = {a} \\n bonus value = {a/100\*5} \\n total salary = {a+(a/100\*5)}")

else:

# 0% bonus

print(f"salary amt = {a} \\n bonus value = 0 \\n total salary = {a}")

else:

print("Invalid Salary")

**Explanation:**

* This code first checks if the salary is a positive number.
* It then uses a series of nested checks to find the correct salary bracket and apply the corresponding bonus percentage.
* The code uses f-strings (f"...") to neatly format the output by embedding variable values directly into the string.

**2. Loops (Repeating Actions)**

Loops are used to execute a block of code repeatedly. This saves you from writing the same code over and over again.

**The range() Function**

The range() function is a powerful tool often used with loops. It generates a sequence of numbers.

* **range(stop)**: Generates numbers from 0 up to (but not including) stop. range(5) -> 0, 1, 2, 3, 4.
* **range(start, stop)**: Generates numbers from start up to stop. range(1, 10) -> 1, 2, 3, 4, 5, 6, 7, 8, 9.
* **range(start, stop, step)**: Generates numbers from start to stop, incrementing by step.

**Important Note:** The range() function itself creates a special range object; it doesn't create a list of numbers directly to save memory. You can convert it to other types like a list or set if needed.

Python

# Convert a range to a list

b = list(range(1,10))

print(b) # Output: [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Convert a range to a set

p = set(range(6))

print(p) # Output: {0, 1, 2, 3, 4, 5}

**Common Error:** The range() function only works with integers. You cannot use strings or characters as its arguments, which will result in a TypeError.

Python

# This code will cause a TypeError because 'a' and 'z' are strings, not integers

w = list(range('a', 'z', 1))

**The for Loop**

The for loop iterates over a sequence (like a list, a string, or a range object) and executes a block of code for each item in that sequence.

**Code Example: Printing Numbers 1 to 10**

Python

# The loop variable 'i' will take on each value from the range, one by one.

for i in range(1,11):

print(i)

**Explanation:**

1. range(1,11) generates the sequence of numbers from 1 to 10.
2. The for loop starts. In the first iteration, the variable i is assigned the first value, 1.
3. The code inside the loop (print(i)) is executed, printing 1.
4. The loop moves to the next value in the range, assigning 2 to i, and prints it.
5. This process repeats until the last number, 10, is printed.

**Controlling the print() function's end character**

By default, print() adds a newline character at the end of its output. You can change this behavior using the end parameter.

Python

for i in range(1,11):

print(i,end=' ')

**Explanation:**

* end=' ' tells the print function to put a space after each number instead of a newline. This causes all the numbers to be printed on the same line, separated by spaces.