Looping - MAD 102 Week 4 Notes

Hia Al Saleh

September 18th, 2024

Contents

1 Looping and Repetition Structures

In programming, repetition structures, also known as loops, are used to execute the same section of code multiple times. This is particularly useful when you want to repeat the same task without duplicating your code, following the DRY principle (Don't Repeat Yourself).

1.1 Advantages of Loops

Loops offer several advantages, including:

- Efficient use of a single set of instructions to process multiple data sets.
- Flexibility to incorporate selection structures, allowing varied treatments of different data based on conditions.
- Ability to operate on known or unknown quantities of data.

1.2 Components of a Loop

A loop has three critical components:

1. **Initialization**: Assign an initial value to the loop variable.
2. **Condition Evaluation**: Determine whether the loop should continue based on a conditional expression.
3. **Alteration**: Modify the loop variable so that the condition will eventually be false, preventing infinite loops.

2 While Loops

A **while loop** executes as long as a specified condition is true. It's a pretest loop, meaning the condition is tested before the loop body is executed.

```
loop_variable =1
while loop_variable <= 3:
  print("loop executed")
  print(f"Lteration:{loop_variable}")
  loop_variable = loop_variable +3

print("loop finished")
```

2.1 For Loops

A **for loop** is commonly used to iterate over a sequence or container (like a list). No sentinel value is required; the loop runs as many times as there are items in the container.

```
list_strings = ['Apple', 'Pear', 'Mango']
for string in list_strings:
    print(string)
```

2.2 Using Ranges in For Loops

The `range()` function allows you to generate a sequence of numbers within a loop.

```
for i in range(5):
    print(i)
for i in range(15, 5, -3):
    print(i)
```

2.3 Ranges

Range	Generated sequence	Explanation
`range(5)`	`0 1 2 3 4`	Every integer from 0 to 4
`range(0, 5)`	`0 1 2 3 4`	Every integer from 0 to 4
`range(3, 7)`	`3 4 5 6`	Every integer from 3 to 6
`range(10, 13)`	`10 11 12`	Every integer from 10 to 12
`range(0, 5, 1)`	`0 1 2 3 4`	Every 1 integer from 0 to 4
`range(0, 5, 2)`	`0 2 4`	Every 2nd integer from 0 to 4
`range(5, 0, -1)`	`5 4 3 2 1`	Every 1 integer from 5 to 1
`range(5, 0, -2)`	`5 3 1`	Every 2nd integer from 5 to 1

2.4 Savings Calculation Example

```
initial_savings = 200
interest_rate = 0.05

years = int(input('Enter years: '))
print()
```

```
savings = initial_savings
for i in range(years):
    print(f'Savings in year {i}: ${savings:.2f}')
    savings += savings * interest_rate
```

2.5 Nested Loops

A **nested loop** is a loop within another loop. The inner loop completes all its iterations for every single iteration of the outer loop.

```
for hours in range(1):
    for minutes in range(60):
        for seconds in range(60):
            print(f'{hours}:{minutes}:{seconds}')
```

3 Break and Continue Statements

The `break` statement allows you to exit a loop prematurely, while the `continue` statement skips the rest of the code in the loop and proceeds to the next iteration.

3.1 Break Statement Example

```
for number in range(1, 101):
    if number % 2 == 1:
        break
    print('This will never run')
```

1. The break statement is used to immediately exit a loop
2. If I want to display the numbers from 1 – 100 that are even, I could use this code.

3.2 Continue Statement Example

```
for number in range(1, 101):
    if number == 5:
        continue
    print(f'{number} is an even number')
```

4 Enumerate Function

Sometimes you need both the index and the value of elements in a collection. The `enumerate()` function creates a tuple with the index and value, allowing easy access to both.

```
students = ["Mal", "Simon", "Zoe", "River"]

for (index, student) in enumerate(students):
    print(f'{student} is located at index {index}')
```

5 Exercises

5.1 Exercise 1

We want to create a program that will add two numbers together and return the result. Unlike previous exercises, we will continue to ask the user for numbers until they no longer want to use our program.

Solution

```
user_choice_to_continue = 'Y'
while user_choice_to_continue == 'Y':
    num1 = int(input('Please enter your first number: '))
    num2 = int(input('Please enter your second number: '))
    sum_result = num1 + num2
    print(f'The sum of {num1} + {num2} = {sum_result}')

    user_choice_to_continue = input('Would you like to
        continue? Y/N: ').upper()
    if user_choice_to_continue not in ['Y', 'N']:
        user_choice_to_continue = input('Invalid input.
            Please enter Y or N: ')
```

5.2 Exercise 2

Create a program that will count down from a specific number to 0.

Solution

```
# introduce the program
print('It\'s a countdown!')
print('==' * 30)
number = int(input('Please enter a number to countdown
    from:'))
# display the results
```

```
print(f'Counting down from {number}:')
while number > -1:
  print(number)
  number -= 1
# end of program
print('Your Program Ends here!')
```

5.3 Exercise 3

Create a countdown program that will count down to 0 from a provided minute value Display the results to the screen

Solution

```
# declare variables
second = 59
# introduce the program
print('It\'s a countdown!')
print('=='*30)
minute = int(input('Please enter a minute to countdown
    from:'))
# display the results
print(f'Counting down from {minute}:')

while minute >=0:
  while second >=0:
    print(f'{minute}:{second}')
    second -=1
  minute -= 1
  second = 59
# end of program
print('Blast off!')
```

5.4 Exercise 4

We want to keep track of our grocery purchases for a day. We then want to know how much we spent. Some days we may not enter anything – so it should display no amount was spent. The purchases vary from day to day

Solution

```
# Declare variables
purchases = []  # Initialize a list to store purchases
total_amount_spent = 0  # Initialize total amount spent

# Introduce Program
```

```
print("Welcome to the Grocery Purchase Tracker!")

# Initial user input
purchased = input("Did you purchase any grocery items
    today ('yes' or 'no'): ").strip().lower()

# Keep prompting the user for purchases until they choose
    to stop
while True:
    if purchased != "yes" and purchased != "no":
        purchased = input("Enter valid input ('yes' or 'no
            '): ").strip().lower()

    if purchased == 'no':
        break
    elif purchased == 'yes':
        item = input("Enter item name: ")

        while True:
            try:
                price = float(input("Enter the price of
                    the item: $"))
                break  # Exit the loop if the price is
                    valid
            except ValueError:
                print("Invalid price. Please enter a valid
                     number.")

        purchases.append((item, price))  # Add the item
            and price to the list of purchases
        total_amount_spent += price  # Update total amount
             spent
        # Ask if more purchases are made
        purchased = input("Did you purchase more grocery
            items today ('yes' or 'no'): ").strip().lower()
# Display the purchases and total amount spent
if purchases:
    print("\nYour purchases for the day:")
    for item, price in purchases:
        print(f"{item}: ${price:.2f}")
    print(f"\nTotal amount spent for the day: ${
        total_amount_spent:.2f}")
else:
    print("\nNo purchases were made today. No amount was
        spent.")
# Sample additional loop example (adjusted)
for i, s in enumerate(range(1, 5)):
    print("index", i)
    print(s)
```