Python Chapter 4: Loops Activities

Contents

Time required: 90 minutes

DRY

Don't Repeat Yourself

Online Tutorials

Go through the following tutorials.

- LearnPython.org Loops
- Python While Loops

    - Python While Loops Exercises 1-4
- Python For Loops
    - Python For Loops Exercises 1-4

for Loop

A **for** loop is a control flow statement used to iterate over a sequence of elements, allowing you to execute a block of code repeatedly for each item in the sequence.

The range() function generates a sequence of numbers.

A for loop is typically used when you know exactly how many times you wish to loop.

```
# range 0 - 9
for i in range(0, 10):
    print(i)
print()
# range 0 - 9, step 2
for i in range(0, 10, 3):
    print(i)
```

- **Sequence:** The sequence can be any iterable object like a list, tuple, string, dictionary, etc. It contains the elements over which the loop iterates.
- **Element**: In each iteration, the for loop assigns the current element from the sequence to the specified variable (in this case, element).
- **Code block**: The indented code block under the for statement is executed for each element in the sequence.

while Loop

A **while** loop is a control flow statement that repeatedly executes a block of code as long as a specified condition is **True**.

A while loop is typically used when you aren't sure how many times you wish to loop.

```python
# Counter variable
count = 0
# While the condition is true
# continue the loop
while count < 10:
    print(count)
    # Increment counter variable
    count += 1
```

break

Break causes the code to exit from the loop early. This is useful in a menu system.

```python
count = 0
while count < 10:
    if (count == 5):
        count += 1
        # break exits the loop
        break
    print(f"{count}")
    count += 1
```

continue

Continue causes the code to go directly to the beginning of the loop.

```python
count = 0
while count < 10:
    if (count % 2 != 0):
        count += 1
        # continue causes the current iteration to
        # go immediately to the beginning of the loop
        continue
    print(count)
    count += 1
```

for else

for and while loops can have an else condition that executes after the last time through the loop.

```
# A Python list
numbers = [1, 2, 3, 4, 5]

for num in numbers:
    if num == 6:
        print("Number 6 found!")
        break
else:
    print("Number 6 not found in the list.")
```

In this example, the loop iterates over the **numbers** list. If the loop finds the number 6, it prints a message and breaks out of the loop. If the loop completes without finding the number 6, the **else** block executes, printing a message indicating that the number was not found in the list.

while else

```
num = 0
while num < 5:
    print(num, "is less than 5")
    num += 1
else:
    print("Loop completed, num is now equal to or greater than 5")
```

In this example, the **while** loop executes as long as the condition **num < 5** is true. Once **num** becomes 5 or greater, the loop stops and the `else` block is executed, printing "Loop completed, num is now equal to or greater than 5".

Input Validation and Exception Handling

Exception handling ensures that the correct data type is input by the user. If the program can’t handle incorrect data, that is an opening for a hacker to get into your system. Many security vulnerabilities are cause by poorly written code.

If the user enters a string when an integer is what the program wants, this will generate a program exception.

Input validation ensures that the input is in the right range. An age should be positive. A bank withdrawal would be negative.

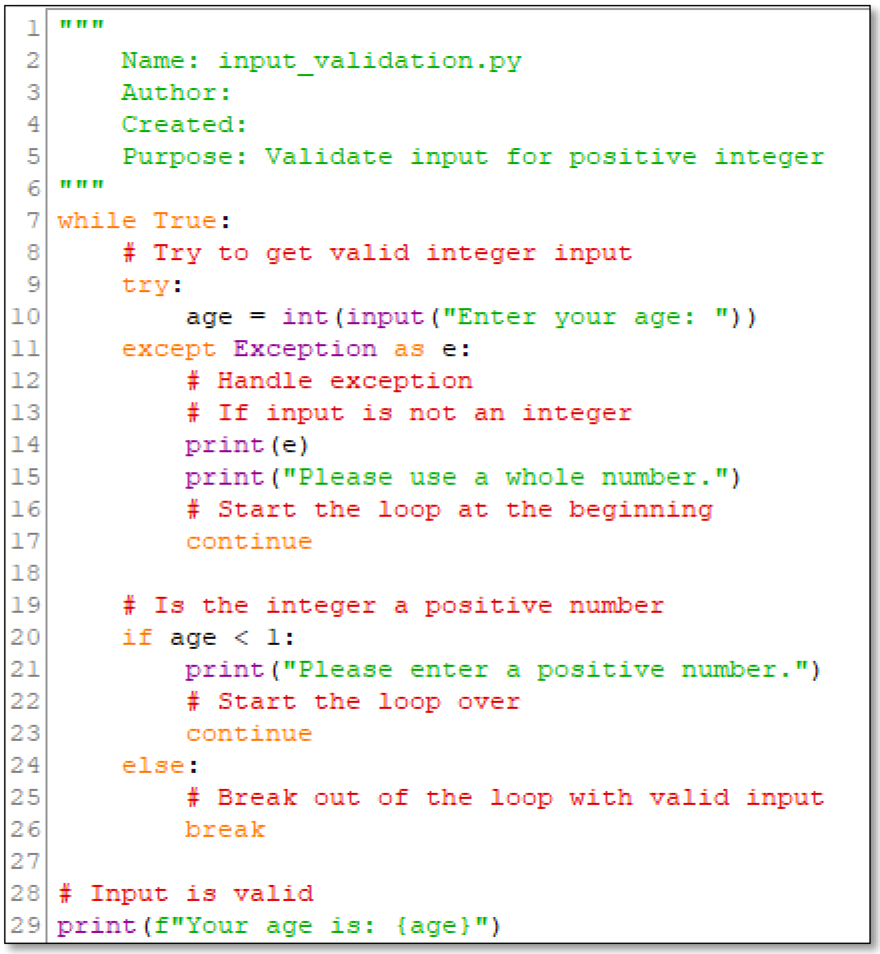
For example, if you want users to enter their ages, your code shouldn’t accept nonsensical answers such as negative numbers (which are outside the range of acceptable integers) or words (which are the wrong data type). Input validation can also prevent bugs or security vulnerabilities.

If you implement a **withdraw_from_account()** function that takes an argument for the amount to subtract from an account, you need to ensure the amount is a positive number. If the **withdraw_from_account()** function subtracts a negative number from the account, the “withdrawal” will end up adding money!

Typically, we perform input validation by repeatedly asking the user for input until they enter valid input, as in the following example:

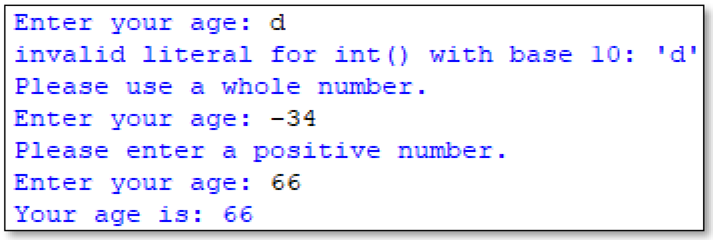
Tutorial 4.1: Input Validation

Create a Python program named **input_validation.py**



```python
"""
    Name: input_validation.py
    Author:
    Created:
    Purpose: Validate input for positive integer
"""
while True:
    # Try to get valid integer input
    try:
        age = int(input("Enter your age: "))
    except Exception as e:
        # Handle exception
        # If input is not an integer
        print(e)
        print("Please use a whole number.")
        # Start the loop at the beginning
        continue

    # Is the integer a positive number
    if age < 1:
        print("Please enter a positive number.")
        # Start the loop over
        continue
    else:
        # Break out of the loop with valid input
        break

# Input is valid
print(f"Your age is: {age}")
```

Example run:

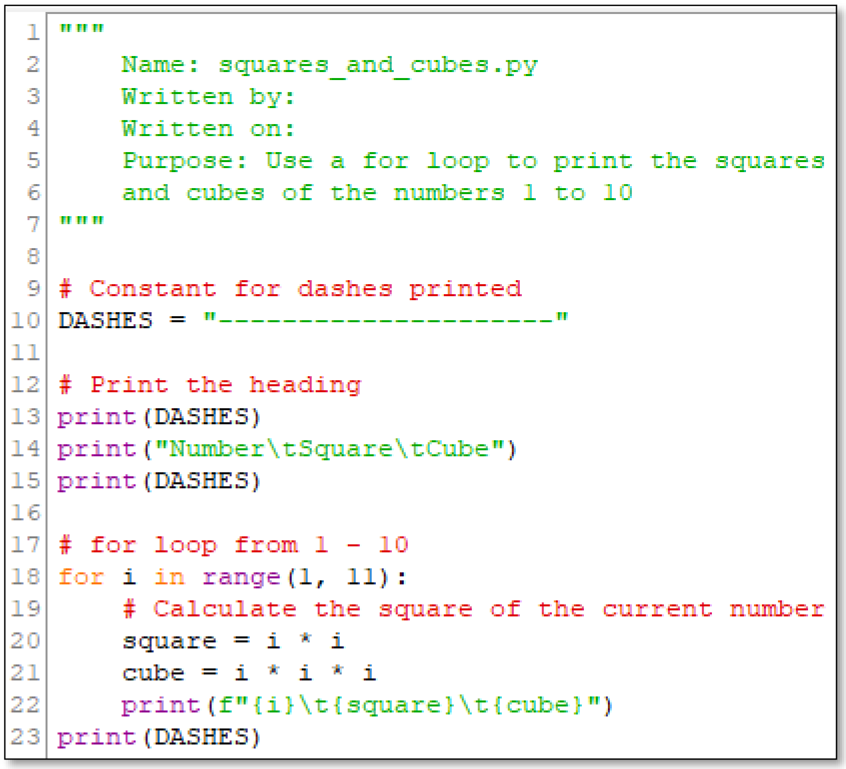


```
Enter your age: d
invalid literal for int() with base 10: 'd'
Please use a whole number.
Enter your age: -34
Please enter a positive number.
Enter your age: 66
Your age is: 66
```

Tutorial 4.2: Squares and Cubes

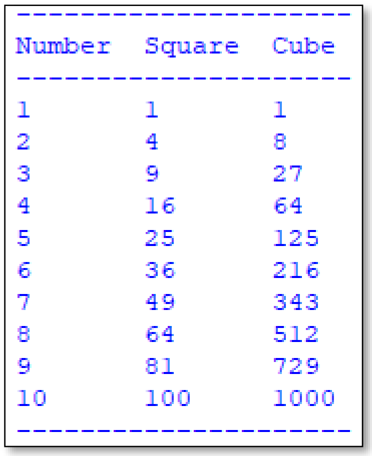
for loops are excellent when we know how many times we need to go through the loop.

The **\t** escape character inserts a tab. It instructs the interpreter to insert a horizontal tab at that point. A horizontal tab is a special whitespace character that typically moves the cursor to the next tab stop.



```python
"""
    Name: squares_and_cubes.py
    Written by:
    Written on:
    Purpose: Use a for loop to print the squares
    and cubes of the numbers 1 to 10
"""

# Constant for dashes printed
DASHES = "----------------------"

# Print the heading
print(DASHES)
print("Number\tSquare\tCube")
print(DASHES)

# for loop from 1 - 10
for i in range(1, 11):
    # Calculate the square of the current number
    square = i * i
    cube = i * i * i
    print(f"{i}\t{square}\t{cube}")
print(DASHES)
```

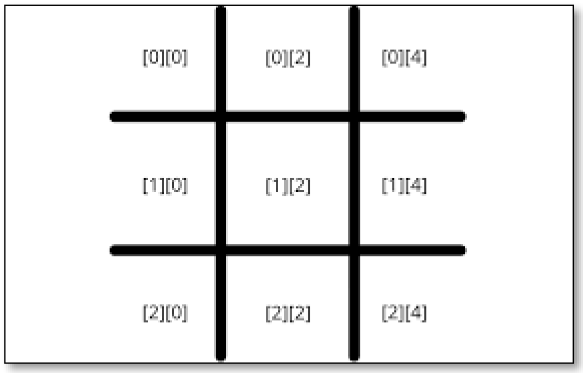
Example run:



```
----------------------
Number   Square   Cube
----------------------
1        1        1
2        4        8
3        9        27
4        16       64
5        25       125
6        36       216
7        49       343
8        64       512
9        81       729
10       100      1000
----------------------
```

Tutorial 4.3: Nested Loops with Tic Tac Toe

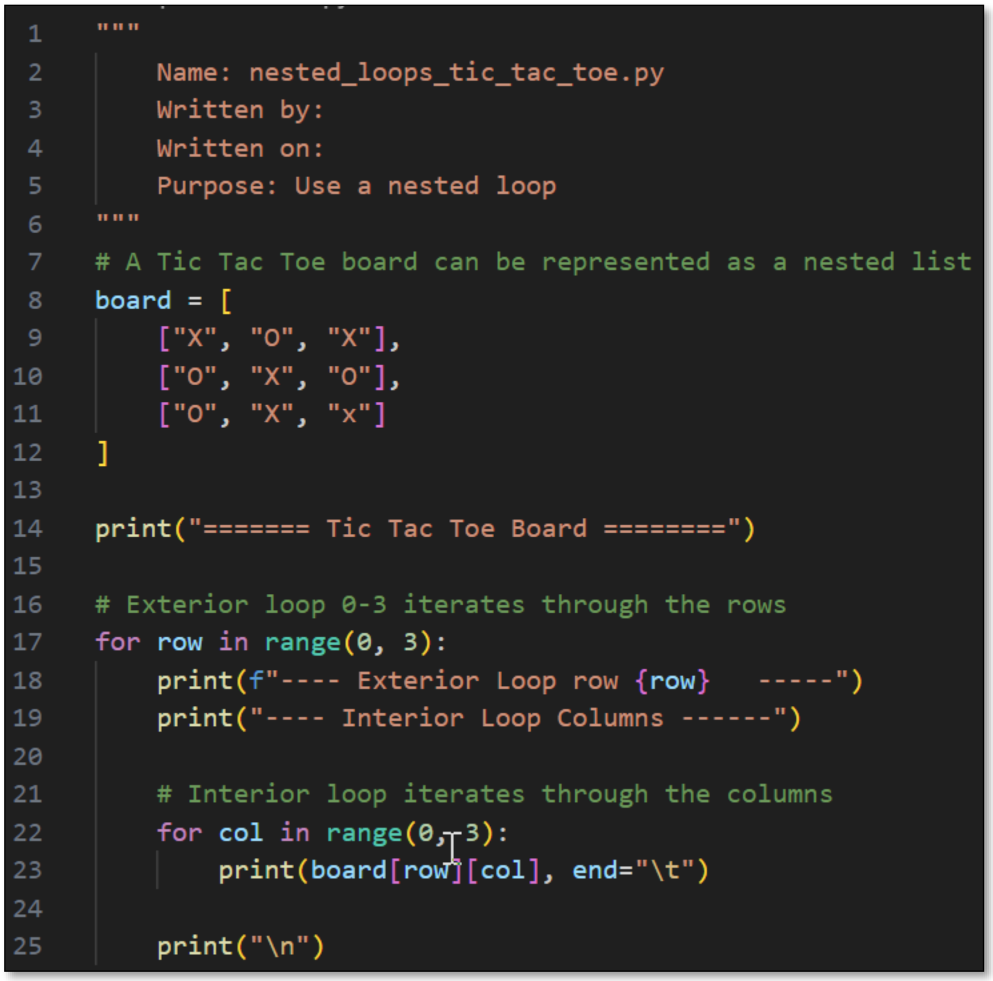
A Tic Tac Toe board is a 2 dimensional 3x3 matrix. In Python this is typically represented as a list of lists.



You can iterate through each cell in the board by using a nested for loop. The outer loop iterates through each row, and the inner loop iterates through each column within that row.

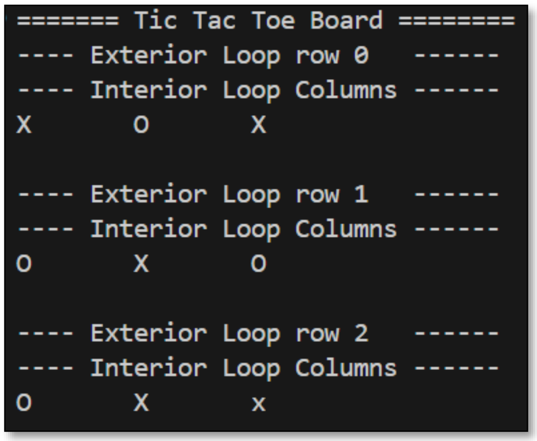
The following program displays the board's current state. Another use case would be to check rows, columns, and diagonals for matching symbols.

Create a Python program named: **tic_tac_toe.py**



```python
"""
    Name: nested_loops_tic_tac_toe.py
    Written by:
    Written on:
    Purpose: Use a nested loop
"""
# A Tic Tac Toe board can be represented as a nested list
board = [
    ["X", "O", "X"],
    ["O", "X", "O"],
    ["O", "X", "x"]
]

print("======= Tic Tac Toe Board ========")

# Exterior loop 0-3 iterates through the rows
for row in range(0, 3):
    print(f"---- Exterior Loop row {row}   -----")
    print("---- Interior Loop Columns ------")

    # Interior loop iterates through the columns
    for col in range(0, 3):
        print(board[row][col], end="\t")

    print("\n")
```

Example run:



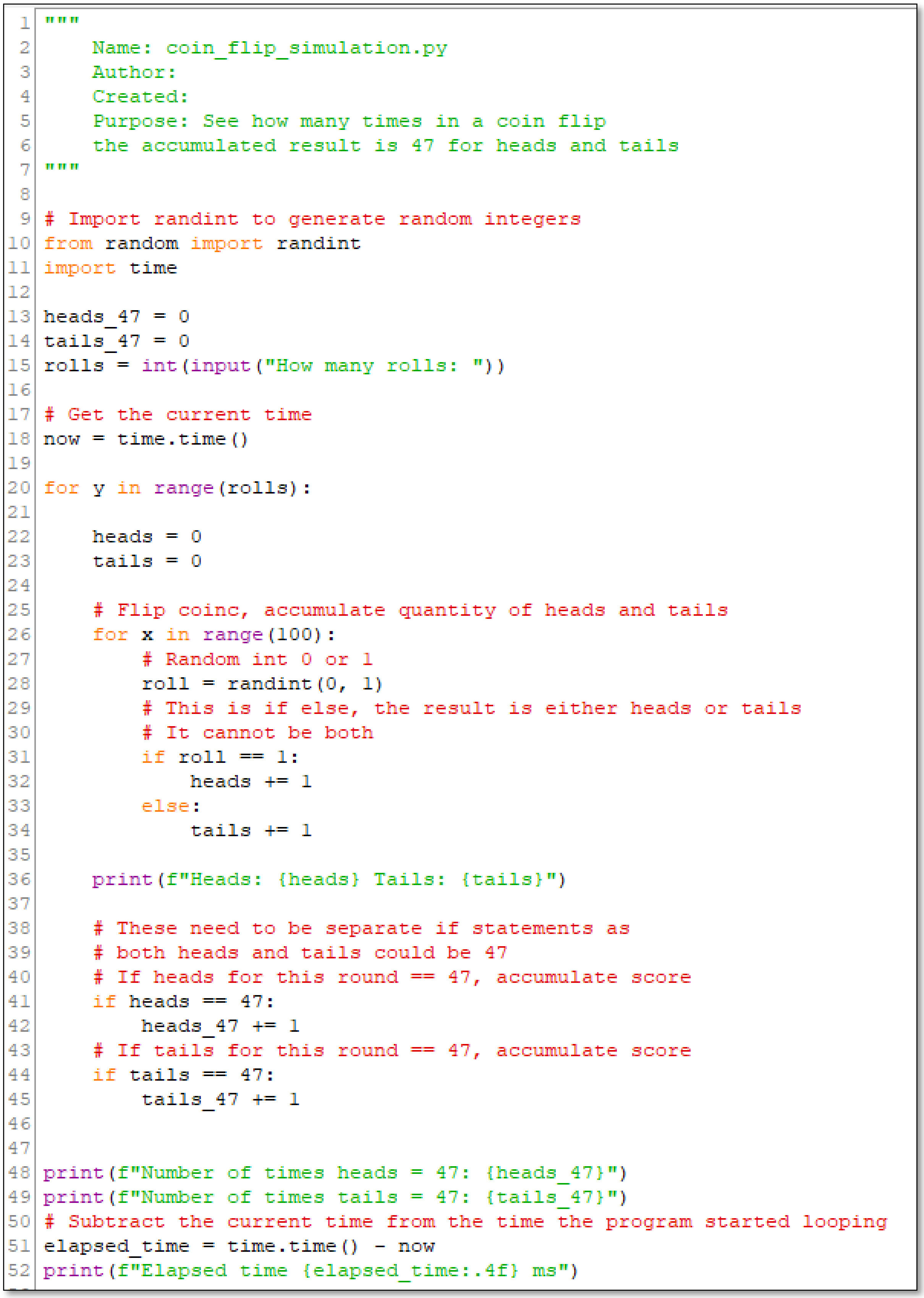
```
======= Tic Tac Toe Board ========
---- Exterior Loop row 0   ------
---- Interior Loop Columns ------
X       O       X

---- Exterior Loop row 1   ------
---- Interior Loop Columns ------
O       X       O

---- Exterior Loop row 2   ------
---- Interior Loop Columns ------
O       X       x
```

Tutorial 4.4: Coin Flip Simulation

If a coin is flipped 1,000 times, accumulate the number of times the result is 47 for heads and tails. Allow the user to determine how many times the program will flip a coin and add up the results.

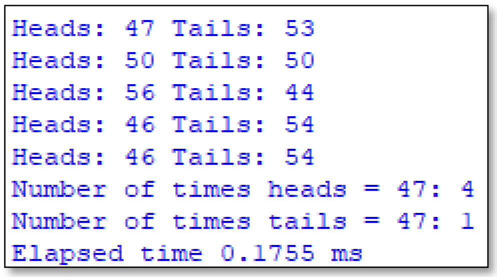
time is a module that is built-in to Python. We use **time.time()** (current computer time down to the ms) to track of how long the program takes to process.

1. Create a Python program named **coin_flip_simulation.py**



```python
"""
    Name: coin_flip_simulation.py
    Author:
    Created:
    Purpose: See how many times in a coin flip
    the accumulated result is 47 for heads and tails
"""

# Import randint to generate random integers
from random import randint
import time

heads_47 = 0
tails_47 = 0
rolls = int(input("How many rolls: "))

# Get the current time
now = time.time()

for y in range(rolls):

    heads = 0
    tails = 0

    # Flip coinc, accumulate quantity of heads and tails
    for x in range(100):
        # Random int 0 or 1
        roll = randint(0, 1)
        # This is if else, the result is either heads or tails
        # It cannot be both
        if roll == 1:
            heads += 1
        else:
            tails += 1

    print(f"Heads: {heads} Tails: {tails}")

    # These need to be separate if statements as
    # both heads and tails could be 47
    # If heads for this round == 47, accumulate score
    if heads == 47:
        heads_47 += 1
    # If tails for this round == 47, accumulate score
    if tails == 47:
        tails_47 += 1


print(f"Number of times heads = 47: {heads_47}")
print(f"Number of times tails = 47: {tails_47}")
# Subtract the current time from the time the program started looping
elapsed_time = time.time() - now
print(f"Elapsed time {elapsed_time:.4f} ms")
```

Example run of 50 coin flips:



```
Heads: 47 Tails: 53
Heads: 50 Tails: 50
Heads: 56 Tails: 44
Heads: 46 Tails: 54
Heads: 46 Tails: 54
Number of times heads = 47: 4
Number of times tails = 47: 1
Elapsed time 0.1755 ms
```

Assignment 1: Iterating Through Numbers

This assignment combines a for loop along with decision making.

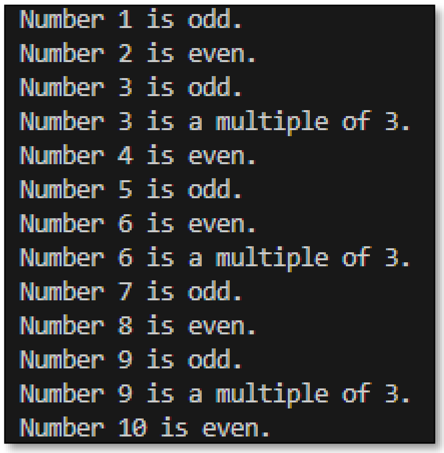
Use a for loop to iterate through a range of numbers and perform specific tasks based on the conditions.

You have been given a task to analyze a range of numbers and perform the following operations:

1. Iterate through the numbers from 1 to 10 (inclusive).
2. For each number, do the following:
    a. If the number is even, print "Number {num} is even."
    b. If the number is odd, print "Number {num} is odd."
    c. If the number is a multiple of 3, print "Number {num} is a multiple of 3."
3. Use TODO and comment your code to indicate your understanding of your solution.

Create a Python program named **number_analysis.py**

Example run:



```
Number 1 is odd.
Number 2 is even.
Number 3 is odd.
Number 3 is a multiple of 3.
Number 4 is even.
Number 5 is odd.
Number 6 is even.
Number 6 is a multiple of 3.
Number 7 is odd.
Number 8 is even.
Number 9 is odd.
Number 9 is a multiple of 3.
Number 10 is even.
```

Challenge

Get input from the user for the range of numbers to be tested.

Assignment Submission

1. Use pseudocode or TODO.
2. Comment your code to show evidence of understanding.
3. Attach the program files.
4. Attach screenshots showing the successful operation of the program.
5. Submit in Blackboard.