ILP 2024 : Computing

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

W1S2
15 May 2024

Objectives

- Introduction to Conditional Statement
  - If
  - Else
  - Elif
  - Nested if/else
- Examples and Practice:
  - Writing simple if statements.
  - Using if-elif-else statements for multiple conditions.
  - Nested if statements.

Conditional Statement

Conditional statements are used to **execute blocks of code** based on **certain conditions.**

```
…… lines of code ……

First_Condition == True
        Block of codes

Second_Condition == True
        Another block of codes
```

Will be executed only when First_Condition is set to be True

Note: Conditional statement allows dynamic control flow in programs.

`if` statement

- The `if` statement is the simplest conditional statement.
- The syntax of the `if` statement in Python consists of the keyword '`if`' followed by a **condition**.
- Add a colon symbol (**:**) after the condition,
- If the condition evaluates to `True`, the **code block** following the `if` statement is **executed**.

```
#checking the if condition

if (variable == True):

    print("Write your code")

#exiting from the if block
```

```
condition = True


if (condition):
    print("I am inside the first 'if' block")


condition = False
if (condition):
    print("I am inside the second 'if' block")
```

```
I am inside the first 'if' block
```

Python indentation

- **Indentation** is used to signify the **beginning** and **ending** of **blocks of code**

- Indentation is not just for **readability**; it is a fundamental aspect of **Python syntax**

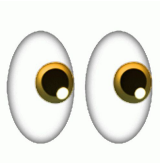
- In Python, **code blocks are defined** by **indentation** rather than curly braces or keywords.

```python
condition = True


if (condition):
    print("I am inside the first 'if' block")


condition = False
if (condition):
    print("I am inside the second 'if' block")
```

```
I am inside the first 'if' block
```

```python
#Example on Python indentation
condition = True

if (condition):
    print("I am inside the first 'if' block")
  print("I am not following indetation")
```

```
  File <string>:6
    print("I am not following indetation")
                                         ^
IndentationError: unindent does not match any outer indentation level
```



`if` with comparison operators

- **Equal to (==)**:
  - Returns True if the operands are equal, otherwise False. Example: x == y
- **Not equal to (!=)**:
  - Returns True if the operands are not equal, otherwise False. Example: x != y
- **Greater than (>)**:
  - Returns True if the left operand is greater than the right operand, otherwise False. Example: x > y
- **Less than (<)**:
- **Greater than or equal** to (>=):
- **Less than or equal to** (<=):

```python
#if with comparison operator

x = 10
y = 5.0
z = "10"


if (x == y):
    print("X and Y are equal")

if (x != y):
    print("X and Y are not equal")

if (x != z):
    print("X and Z are not equal")
```

```
X and Y are not equal
X and Z are not equal
```

Programming test - 1

1. Ask user to enter three numbers and find the maximum number among them.

```
Enter the first number:  23
Enter the second number:  34
Enter the third number:  45
The maximum number among 23 , 34 , and 45 is: 45
```

2. Check if a given year is a leap year or not.

3. Determine if a given number is even or odd.

else statement

Can you tell the output of the below code ?

```
condition = False
if (not condition):
    print("I am inside the second 'if' block")
```

else

```
#Else condiotional statement
condition = 0

if (condition):
    print("I am inside the first 'if' block")

else:
    print("I am inside the 'else' block")
```

```
I am inside the 'else' block
```


Note: **No Boolean condition** to be checked for `else` statement

Programming test - 2

Change the previous code using `if` and `else` statements

1. Ask user to enter three numbers and find the maximum number among them.

```
Enter the first number:  23
Enter the second number:  34
Enter the third number:  45
The maximum number among 23 , 34 , and 45 is: 45
```

2. Check if a given year is a leap year or not.

3. Determine if a given number is even or odd.

Contd…

Ask the user to input his/her age, and determine whether the user is a minor, an adult, or a senior citizen based on the following conditions.

- If the age is less than 18, we print "You are a minor."
- If the age is between 18 (inclusive) and 65 (exclusive), we print "You are an adult."
- If the age is 65 or greater, we print "You are a senior citizen."

```python
#elif statement

# Ask the user to enter their age
age = int(input("Enter your age: "))

# Check the age range and print a message accordingly
if age < 18:
    print("You are a minor.")
if age >= 18 and age < 65:
    print("You are an adult.")
else:
    print("You are a senior citizen.")
```

```
Enter your age:  17
You are a minor.
You are a senior citizen.
```

Note: `else` statement only checks the condition of the immediate preceding `if` statement .

elif statement

- The **elif** (short for “else-if”) statement is used when we have **multiple conditions** to check.
- **elif** allows you to check additional conditions after the initial **if** statement.

```python
#elif statement

# Ask the user to enter their age
age = int(input("Enter your age: "))

# Check the age range and print a message accordingly
if age < 18:
    print("You are a minor.")
elif age >= 18 and age < 65:
    print("You are an adult.")
else:
    print("You are a senior citizen.")
```

```
Enter your age:  17
You are a minor.
```

Can you quickly write a program to check whether an entered number is greater than, equal to, or less than 5? Do it now

```python
# Test
x = 10

if x > 5:
    print("x is greater than 5")
elif x == 5:
    print("x is equal to 5")
else:
    print("x is less than 5")
```

```
x is greater than 5
```

Contd…

Can you guess the output ?

```
var1 = True
var2 = True

if (var1):
    print("Printing from 'if' block")

elif (var2):
    print("Printing from 'elif' block")

else:
    print("Printing from 'else' block")
```

```
Printing from 'if' block
```

Note 1: If the Boolean in the `if` statement is **True**, execute the code inside the `if`, ignore the rest of the `elif` statement(s).

Note 2: `else` comes last, after all the `elif/if` statements

Programming Test -3

1. Write a Python program to determine the letter grade for a student based on their percentage score. The program should prompt the user to input their percentage score, and then output their corresponding letter grade according to the following criteria:

A: 90% or above

B: 80% to 89%

C: 70% to 79%

D: 60% to 69%

F: Below 60%

Your program should use `if`, `elif`, and `else` statements.

```python
# Ask the user to enter their grade
grade = int(input("Enter your grade (out of 100): "))

# Determine the grade level
if grade >= 90:
    print("Your grade is A.")
elif grade >= 80:
    print("Your grade is B.")
elif grade >= 70:
    print("Your grade is C.")
elif grade >= 60:
    print("Your grade is D.")
else:
    print("Your grade is F. You need to improve.")
```

```
Enter your grade (out of 100):  85
Your grade is B.
```

2. Can you write the same program using only `if` statement?

Dead code

Dead code: we call “dead code” a piece of code that was written, but is **never going to be executed**. Often, due to **bad logic** in code.

Be careful!


```python
if(x > 10):
    print("Hello!")
elif(x > 12):
    print("World!")
```

Can you find the dead code?

Nested conditional statements

- **Nested conditional statements** are the statements **within other conditional statement**.
- They allow for more **complex decision-making** by checking multiple conditions within different levels of indentation.

```python
# Prompt the user to enter their age
age = int(input("Enter your age: "))

# Nested if/else statements to determine user's age group
if age < 0:
    print("Invalid age. Age cannot be negative.")
else:
    if age < 18:
        print("You are a minor.")
    elif age < 65:
        print("You are an adult.")
    else:
        print("You are a senior citizen.")
```

```
Enter your age:  19
You are an adult.
```

Programming Test -4

Write a Python program to ask user to enter a number.

- If the number is positive, determine if the entered number is odd or even.
- If the number is odd, check whether it is divisible by either 5, 3, or both.

```python
# Prompt the user to enter a number
num = int(input("Enter a number: "))

# Check if the entered number is positive
if num > 0:
    # Determine if the number is odd or even
    if num % 2 != 0:
        print("The entered number is odd.")

        # Check if the number is divisible by 5, 3, or both
        if num % 3 == 0 and num % 5 == 0:
            print("The number is divisible by both 3 and 5.")
        elif num % 3 == 0:
            print("The number is divisible by 3.")
        elif num % 5 == 0:
            print("The number is divisible by 5.")
        else:
            print("The number is not divisible by either 3 or 5.")
    else:
        print("The entered number is even.")
else:
    print("Error: Please enter a positive number.")
```

```
Enter a number:  15
The entered number is odd.
The number is divisible by both 3 and 5.
```

Contd…

Can you guess the output?

```python
# Prompt the user to enter a number
num = int(input("Enter a number: "))

# Check if the entered number is positive
if num > 0:
    # Determine if the number is odd or even
    if num % 2 != 0:
        print("The entered number is odd.")

        # Check if the number is divisible by 5, 3, or both
        if num % 3 == 0:
            print("The number is divisible by 3.")
        elif num % 3 == 0 and num % 5 == 0:
            print("The number is divisible by both 3 and 5.")
        elif num % 5 == 0:
            print("The number is divisible by 5.")
        else:
            print("The number is not divisible by either 3 or 5.")
    else:
        print("The entered number is even.")
else:
    print("Error: Please enter a positive number.")
```

```
Enter a number:  15
```

Note: Always be careful with **conditional statements**. They are often overly complicated and prone to errors in designing the code.

Contd…

To reduce complexity of your code, sometimes, you can convert a nested **`if-else`** blocks to only **`if`** blocks.

```python
x = 5
if(x==0):
    print("The number x is zero.")
elif(x>=0):
    print("The number x is positive.")
    if(x>0):
        print("In fact, the number x is STRICTLY positive.")
else:
    print(("The number x is negative."))
    if(x<0):
        print("In fact, the number x is STRICTLY negative.")
```

```
The number x is positive.
In fact, the number x is STRICTLY positive.
```

```python
x = 5
if(x==0):
    print("The number x is zero.")
if(x != 0 and x>=0):
    print("The number x is non-zero and positive.")
if(x>0):
    print("In fact, the number x is STRICTLY positive.")
if(x != 0 and x<=0):
    print(("The number x is non-zero and negative."))
if(x<0):
    print("In fact, the number x is STRICTLY negative.")
```

```
The number x is non-zero and positive.
In fact, the number x is STRICTLY positive.
```

```
while
statement
```

Exercise

Write a program that checks if **x** is less than 5 using only **if** statements and prints the number, starting from 0

```
magic_code
    if x < 5:
        print(x)
    x += 1
Goto magic_code
```

```python
#Introduction to While loop
'''
Write a program that checks if x is less than 5 using only
if statements and prints the number, starting from 0

'''

x = 0

if x < 5:
    print(x)
x = x + 1

if x  < 5:
    print(x)
x += 1

if x  < 5:
    print(x)
x += 1

if x  < 5:
    print(x)
x += 1

if x  < 5:
    print(x)
x += 1
```

```
0
1
2
3
4
```

`while` statement

The `while` statement is another type of **conditional statement**.

```
magic_code
    if x < 5:
    print(x)
    x += 1
    Goto magic_code
```

→

```
while(x < 5):
    print(x)
    x += 1
```

How it works:

- If the Boolean condition specified for the `while` statement is **True**, then **execute** the block of code inside the while statement.
- If the Boolean condition is **False**, **ignore** the block of code in the while statement.

Contd …

```python
#Introduction to While loop
'''
Write a program that checks if x is less than 5 using only
if statements and prints the number, starting from 0

'''

x = 0

if x < 5:
    print(x)
x = x + 1

if x  < 5:
    print(x)
x += 1

if x  < 5:
    print(x)
x += 1

if x  < 5:
    print(x)
x += 1

if x  < 5:
    print(x)
x += 1
```

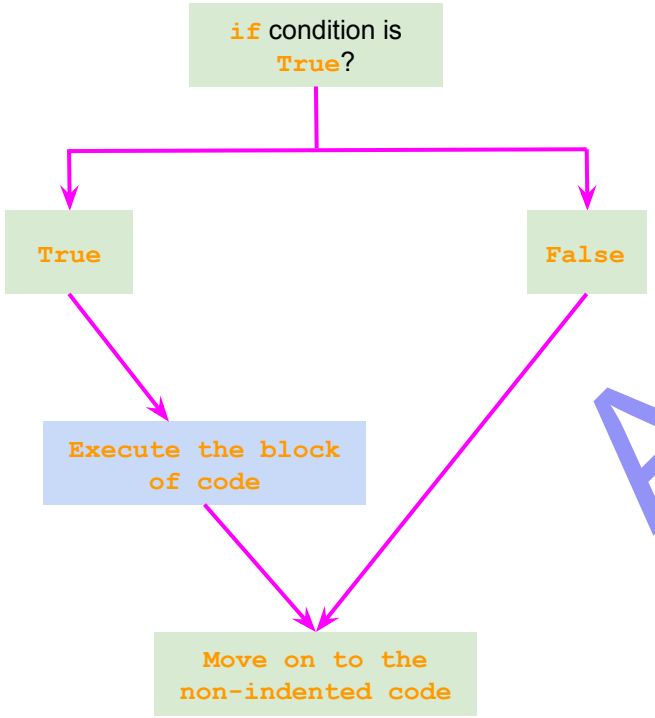
```
0
1
2
3
4
```

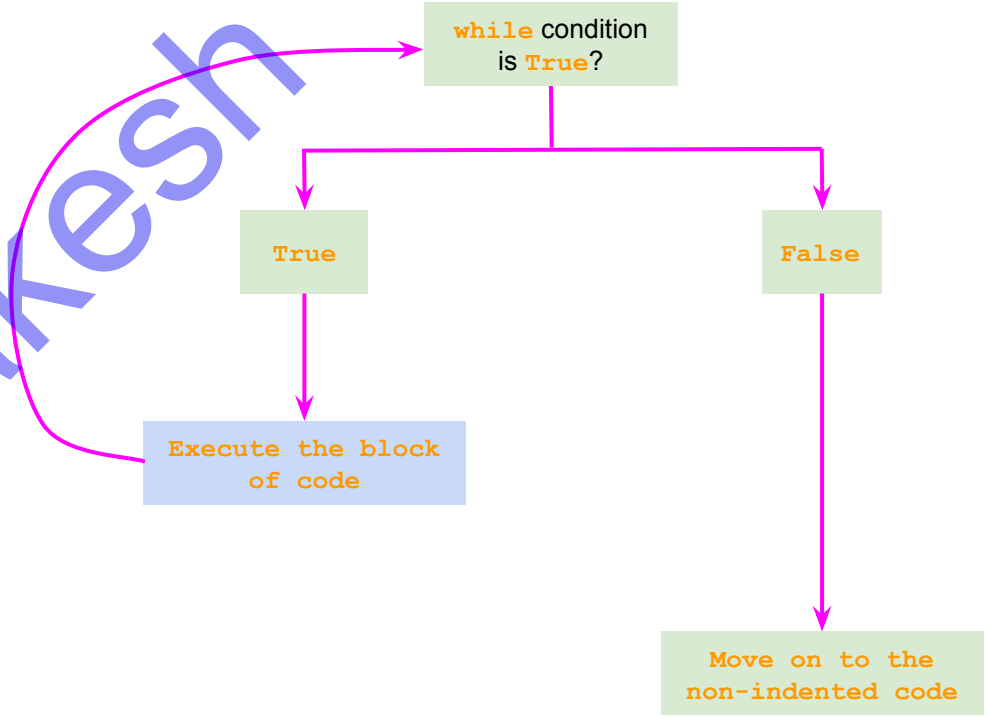
```python
#using while statement

x = 0

while (x<5):
    print(x)
    x += 1

print("Exiting from 'while' loop")
```

```
0
1
2
3
4
Exiting from 'while' loop
```

```python
while (x<5 and x >= 0):
```

if Vs while





Programming test – 4

Write a program to ask user a number and print the multiplication table of that number using **while** loop/statement

```
Enter a number:  7
Multiplication Table of 7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

```python
num = int(input("Enter a number: "))
i = 1

print("Multiplication Table of", num)

while i <= 10:
    #print(num, "x", i, "=", num * i)
    print (f"{num} x {i} = {num * i}")
    i += 1
```

Programming test – 5

Write a program that asks the user for a positive integer and calculates its factorial using a while loop.

```python
#Write a program that asks the user for a positive integer and calculates its factorial using a while loop.

num = int(input("Enter a positive integer: "))  # Ask the user for a positive integer
factorial = 1
i = 1

# Check if the input is a positive integer
if num < 0:
    print("Factorial is not defined for negative numbers.")
elif num == 0:
    print("Factorial of 0 is 1.")
else:
    while i <= num:
        factorial *= i
        i += 1
    print("Factorial of", num, "is", factorial)
```

```
Enter a positive integer:  5
Factorial of 5 is 120
```

Programming test – 6

Can you create a computer virus?

Hint: infinite while loop :) something related to **condition**

```python
#virus problem

x = 1

while (True):
    print("Virus", x)
    x += 1

print("Exited from the 'while' loop")
```

```
Virus 195615
Virus 195616
Virus 195617
Virus 195618
Virus 195619
Virus 195620
Virus 195621
Virus 195622
```

```
IOPub data rate exceeded.
The Jupyter server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--ServerApp.iopub_data_rate_limit`.

Current values:
ServerApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
ServerApp.rate_limit_window=3.0 (secs)
```

Infinite loops

The statements inside the **`while`** repeat till the condition is **`True`**.

Is there any solution for infinite loop?

- Ctrl +Z
- CMD + C (MacBook)
- Stop button on Jupyter

Is there any coding approach to stop infinite loop?

Ans: **`break`** statement

```
#virus problem with break statment

x = 1

while (True):
    print("Virus", x)
    x += 1
    if x > 10:
        break
print("Exited from the 'while' loop")
```

```
Virus 1
Virus 2
Virus 3
Virus 4
Virus 5
Virus 6
Virus 7
Virus 8
Virus 9
Virus 10
Exited from the 'while' loop
```

`break` statement

- The `break` statement provides a crucial mechanism for **controlling the flow of loops**.
  - **Premature Termination:** Allows for the immediate exit from loops.
  - **Conditional Break:** Executes based on specific conditions.

Programming test -7

Design a guessing game where user has to guess a secret number between 1 and 5. Use the `break` statement to exit the game when the correct number is guessed or when the user decides to *quit*.

Instructions:

- `while` loop that continues until the user decides to *quit*.
- Within the loop, generate a new secret number between 1 and 5 for each attempt.
- Prompt the user to guess the number.
- Compare the user's guess with the secret number.
- Use the `break` statement to exit the loop when the *correct number is guessed* or when the user decides to *quit*.

Hint:

```
import random
secret_number = random.randint(1, 5)
```

```
Welcome to the Guessing Game!
Try to guess the secret number between 1 and 5.
Enter your guess (or 'quit' to exit):  2
The secret number is 4
Enter your guess (or 'quit' to exit):  1
The secret number is 2
Enter your guess (or 'quit' to exit):  1
The secret number is 4
Enter your guess (or 'quit' to exit):  1
Congratulations! You guessed the correct number 1 in 4 attempts.
```

```
Welcome to the Guessing Game!
Try to guess the secret number between 1 and 5.
Enter your guess (or 'quit' to exit):  quit
Goodbye! Thanks for playing.
```

Cont …

```python
import random

print("Welcome to the Guessing Game!")
print("Try to guess the secret number between 1 and 5.")

attempts = 0

while True:
        secret_number = random.randint(1, 5)
        guess = input("Enter your guess (or 'quit' to exit): ")

        # Check if the user wants to quit
        if guess == 'quit':
            print("Goodbye! Thanks for playing.")
            break

        # Validate input
        if not guess.isdigit():
            print("Please enter a valid number.")
            continue

        guess = int(guess)
        attempts += 1

        # Check if guess is correct
        if guess == secret_number:
            print(f"Congratulations! You guessed the correct number {secret_number} in {attempts} attempts.")
            break
        else:
            print("The secret number is",secret_number)
```

Infinite loop

Can you spot the differences?

```
#virus problem with break statment

x = 1

while (True):
    print("Virus", x)
    x += 1
    if x > 10:
        break
print("Exited from the 'while' loop")
```

```
#virus problem with break statment

x = 1

while x < 10:
    print("Virus", x)
    x += 1
print("Exited from the 'while' loop")
```


It is often easily avoided, by using the **Boolean** expression of the `if` statement used for **break**, as the **condition** in the `while` statement

do - while loop

In Python, do - while loop is not available, but in other programming languages like C, C++, Java, it is available.

```
do{

….

….

}while(condition)
```

The do-while block will be executed at least once.

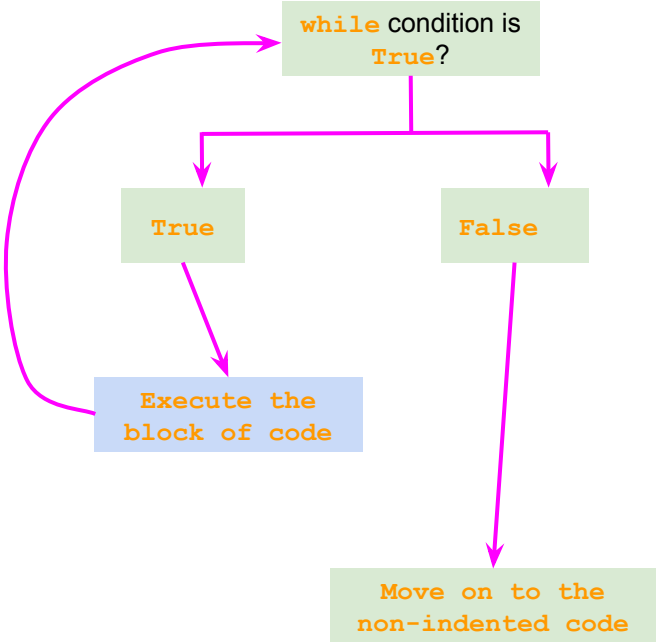
for loop

Initialize

Validation

Increment

```
num = int(input("Enter a number: "))
i = 1

print("Multiplication Table of", num)

while i <= 10:
    #print(num, "x", i, "=", num * i)
    print (f"{num} x {i} = {num * i}")
    i += 1
```

for loop **combines** them together

`for` loop

`for` initialization; condition; increment

…

…

next statements

```
for i in range(0,5):
    print("Virus", i)

print("Exited from the 'for' loop" )
```

```
Virus 0
Virus 1
Virus 2
Virus 3
Virus 4
Exited from the 'for' loop
```

- `range(5)` generates a sequence of numbers from **0 to 4**.
- The `for` loop iterates over each value of `i` in the generated sequence.
- The loop **body** (the indented block of code) is executed for each value of `i`.

Programming test -8

Write a program to ask user a number and print the multiplication table of that number using **for** loop/statement

```
Enter a number:  7
Multiplication Table of 7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

```python
#Write a program to print the multiplication table of 7 using for loop/statement


num = int(input("Enter a number: "))  # Ask the user for a number
i = 1

print("Multiplication Table of", num)

for ? in range (1,???):
    print (f"{num} x {i} = {num * i}")
    i += 1
```

```
Enter a number:  7
Multiplication Table of 7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

`range()` function

- The `range()` function in Python is used for generating **sequences of numbers.**

- Syntax: `range(start, stop, step)`

  Example: `range(5)` generates numbers from 0 to 4.

  `range(0,10,2)` generates numbers: 0, 2, 4, 6, 8

  `range(5, 0, -1) ???` 5, 4, 3, 2, 1

```
for i in range(3):
    print(i)
```

Note: default value of **start** is 0

Infinite loop using `for`

Not possible using `range ()`

Exercises (using `for` loop)

1. Write a Python program to calculate the sum of all numbers from 1 to n
2. Print the multiplication tables of all the numbers 1-10 using nested `for` loop

```
Multiplication Table of 1
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9
1 * 10 = 10

Multiplication Table of 2
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
```

Exercises (using `for` loop)

Write a Python program to print the below pattern using nested `for` **loops**.

```
*
**
***
****
*****
******
*******
********
*********
```

Hint: Need two `for` loops; a string which holds *

```python
for i in range (1,10):
    x ="*"
    for j in range(1,i):
        x += "*"
    print(x)
```

```
*
**
***
****
*****
******
*******
********
*********
```

Contd …

- The statement `print("*", end=" ")` in Python is used to print an asterisk (*) **without moving to a new line**.

- When you write `print("*")`, it's equivalent to writing `print("*", end="\n")`.

- **By default**, the `end` parameter of the `print()` function is set to `"\n"`, newline character.

```python
# Printing a pattern using nested for loops
for i in range(10):  # Outer loop for rows
    for j in range(i + 1):  # Inner loop for column
        print("*", end=" ")  # Print a star
    print()  # Move to the next line after each row
```

```
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
* * * * * * * * * *
```

Practice

- Can you draw a square

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

- Can you draw the below pattern

```
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```

n = 5

```
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```

n = 6

Practice

Can you draw the below pattern

```
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```

n = 5

```python
# Printing a pattern using nested for loops
n = 5
for i in range(1, n + 1):  # Outer loop for rows
    for j in range(n - i):  # Inner loop for spaces
        print(" ", end=" ")  # Print space
    for k in range(2 * i-1):  # Inner loop for stars
        print("*", end=" ")  # Print a star
    print()  # Move to the next line after each row
```

```
    *
  * * *
* * * * *
* * * * * * *
* * * * * * * * *
```

- Need a **for** loop to print stars * for each raw
  - Need a **for** loop to print **spaces**
    – It calculates the number of spaces needed based on the **row number** (**i**) and the **total number of rows (n)**.

```
0 1 2 3 *
0 1 2 * * *
0 1 * * * * *
0 * * * * * * *
* * * * * * * * *
```

```python
for j in range(n - i):
        print(" ", end=" ")
```

- Need a **for** loop to print stars in each row follows the pattern: 1, 3, 5, 7, ..., (2 * i - 1).
- **print()** ---> new line

break and continue statements

break

```python
# Using a for loop with a break statement
for i in range(1, 11):
    if i == 6:  # Check if the current value of i is 6
        break  # If i is 6, exit the loop immediately
    print(i)  # Print the current value of i

print("Exited from the 'for' loop")
```

```
1
2
3
4
5
Exited from the 'for' loop
```

continue

```python
# Using a for loop with a break statement
for i in range(1, 11):
    if i == 6:  # Check if the current value of i is 6
        print("Skipping number 6")
        continue  # If i is 6, exit the loop immediately
    print(i)  # Print the current value of i

print("Exited from the 'for' loop")
```

```
1
2
3
4
5
Skipping number 6
7
8
9
10
Exited from the 'for' loop
```

for loop with exit

Can you write a program to check whether an entered number is prime of not.

```
# Checking if a number is prime
num = int(input("Enter a number"))

for i in range(2, num):
    if num % i == 0:
        print(f"{num} is not a prime number.")
        break
else:
    print(f"{num} is a prime number.")
```

```
Enter a number 6
6 is not a prime number.
```


Note: `break` statement also skips the `else` statement

Programming test -8

Design a guessing game where user has to guess a secret number between 1 and 5. Use the `break` statement to exit the game when the correct number is guessed or when the user decides to *quit*.

Instructions:

- `for` with `else` that continues until the user decides to *quit*.
- Within the loop, generate a new secret number between 1 and 10 for each attempt.
- Prompt the user to guess the number.
- Compare the user's guess with the secret number.
- Use the `break` statement to exit the loop when the *correct number is guessed* or when the user decides to *quit*.
- **Maximum number of attempt is 5** (new rule)

Hint: `import random`

```
secret_number = random.randint(1, 5)
```

```
Welcome to the Guessing Game!
Try to guess the secret number between 1 and 10.

Attempt 1/5: Enter your guess:  1
Try again! Secret number was 2

Attempt 2/5: Enter your guess:  1
Try again! Secret number was 2

Attempt 3/5: Enter your guess:  1
Try again! Secret number was 6

Attempt 4/5: Enter your guess:  1
Try again! Secret number was 9

Attempt 5/5: Enter your guess:  1
Try again! Secret number was 3

Sorry, you've exhausted all attempts. The correct number was 3.
```