unit-2

Objectives

- There are situations in our lives when we need to make a decision and take the next steps accordingly. Similar situations arise in a programming language, and we need to make decisions here as well. Based on the decision, we execute the next block of code.

if statement

if..else statement

elif statements

Nested if statements

if statement

- In this, the statement starts with an if reserved word. The condition in the statement is a Boolean expression that determines whether or not the body will be executed. A colon must follow the condition. The block is defined as one or more statements that are executed when the condition is true. The statement in the if block is a Boolean expression, which determines whether or not the block of statements will be executed.

```
Syntax:
if expression:
    statement(s)
```

In [2]:
```
n= int(input("enter value: "))
if n%2==0:
 print("number is even")
```

```
enter value: 5
```

In [3]:
```
n= int(input("enter value: "))
if n%2==0:
 print("number is even")
```

```
enter value: 10
number is even
```

if..else statement

- With if else statements, if is the reserved word, and the code block under it is executed if its boolean expression evaluates to TRUE; otherwise, the code block under the else statement is executed.

Syntax:

if expression:

    statement(s)

else:

    statement(s)

In [4]:
```
n= int(input("enter value: "))
if n%2==0:
 print("number is even")
else:
 print("number is odd")
```

```
enter value: 1
number is odd
```

elif statements

- The elif statement helps you evaluate multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.

Syntax:

if expression1:

    statement(s)

elif expression2:

    statement(s)

elif expression3:

    statement(s)

else:

    statement(s)

In [5]:
```
n=int(input("enter value: "))
if n>0:
 print("posative value")
elif n<0:
 print("negative value")
else:
 print("value is zero")
```

```
enter value: -2
negative value
```

Nested if statements

- We require conditions inside conditions sometimes, and we use if under if or if…elif…else under if…elif…else or other nested conditions at such times.

- Syntax:

if expression1:

    statement(s)

if expression2:

    statement(s)

elif expression3:

    statement(s)

else

    statement(s)

elif expression4:

    statement(s)

else:

    statement(s)

In [6]:
```
year=int(input("enter year: "))
if year%4==0:
 if year%100!=0:
     print("leap year")
 elif year%400==0:
     print("leap year")
 else:
     print("not leap year")
else:
 print("not leap year")
```

```
enter year: 1900
not leap year
```

In [7]:

```
# min of 3 number

a = int(input('first num'))
b = int(input('second num'))
c = int(input('third num'))

if a<b and a<c:
  print('smallest is',a)
elif b<c:
  print('smallest is',b)
else:
  print('smallest is',c)
```

```
first num5
second num7
third num9
smallest is 5
```

In [8]:
```
print("Options:")
print("Enter 'add' for addition")
print("Enter 'subtract' for subtraction")
print("Enter 'multiply' for multiplication")
print("Enter 'divide' for division")


user_input = input(": ")

if user_input == "add":
    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))
    result = num1 + num2
    print("Result:", result)
elif user_input == "subtract":
    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))
    result = num1 - num2
    print("Result:", result)
elif user_input == "multiply":
    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))
    result = num1 * num2
    print("Result:", result)
elif user_input == "divide":
    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))
    if num2 == 0:
        print("Cannot divide by zero")
    else:
        result = num1 / num2
        print("Result:", result)
else:
    print("Invalid input")
```

```
Options:
Enter 'add' for addition
Enter 'subtract' for subtraction
Enter 'multiply' for multiplication
Enter 'divide' for division
: add
Enter first number: 5
Enter second number: 7
Result: 12.0
```

Modules in Python

- random
- math

In [9]:
```
# math
import math

math.sqrt(196)
```

Out[9]: 14.0

In [10]:
```
# random
import random
print(random.randint(1,100))
```

```
94
```

In [11]:
```
# Guessing game

# generate a random integer between 1 and 100
import random
jackpot = random.randint(1,100)

guess = int(input('guess karo'))
counter = 1
while guess != jackpot:
  if guess < jackpot:
    print('galat!guess higher')
  else:
    print('galat!guess lower')

  guess = int(input('guess karo'))
  counter += 1

else:
  print('correct guess')
  print('attempts',counter)
```

```
guess karo15
galat!guess higher
guess karo35
galat!guess higher
guess karo75
galat!guess higher
guess karo90
galat!guess lower
guess karo85
correct guess
attempts 5
```


Loops in Python

- Need for loops
- While Loop
- For Loop

- Generally, statements execute in sequence. However, you may sometimes need to run a block of code several times, and you can use loops at such times. Loops are statements that repeat an action over and over The control flow goes to the loop's body and executes the statement if the Boolean expression is True in the preceding figure; otherwise, it exits from the loop.

While loop

- Python supports the while loop, which is used to iterate over a block of code as long as the test expression (condition) is true. We generally use this loop when we don't know the

number of times to iterate beforehand

while expression>:

```
# as long as the expression evaluates to True

block of code>
```

In [12]:

```
x=1
while x<10:
 print(x,"hello")
 x+=1

```

```
1 hello
2 hello
3 hello
4 hello
5 hello
6 hello
7 hello
8 hello
9 hello
```

For loop

- The for loop is a common iterator in Python. It can step through the items in an ordered sequence or other iterable objects. The for loop statement supports strings, lists, tuples, and other built-in iterables as well as new user-defined objects.

Syntax:

For iterating_var in sequence: #(sequence mean range(),list[],string,tuple,dictionary values and keys)

```
Statements(s)
```

In [13]:
```
#find the first n term sum
sum=0
n=int(input("enter number:"))
'''1 for starting range
n+1 for ending range it is not include last 1 denote the increment or decr
for i in range(1,n+1,1):
 sum+=i
 print(i)
print("total sum is: ",sum)
```

```
enter number:10
1
2
3
4
5
6
7
8
9
10
total sum is:  55
```

In [14]:
```
#find the first n even term sum
sum=0
n=int(input("enter number:"))
'''1 for starting range
n+1 for ending range it is not include
last 1 denote the increment or decrement'''
for i in range(2,2*n+1,2):
 sum+=i
 print(i)
print("total sum is: ",sum)
```

```
enter number:5
2
4
6
8
10
total sum is:  30
```

range(begin,end,step)

- Here, begin is the initial value given in the range; the default value becomes zero if this is not included.
- end is the value that comes after the last value in the range; the end value is not deleted.
- step indicates the amount to increment or decrement; it defaults to 1 (increments by one) if the change parameter is omitted.
- The values in begin, end, and step must be integer values; floating-point values and other types are not allowed

Sequence sum

1/1! + 2/2! + 3/3! + ...

In [15]:
```
# Code here

n = int(input('enter n'))

result = 0
fact = 1

for i in range(1,n+1):
  fact = fact * i
  result = result + i/fact

print(result)
```

```
enter n5
2.708333333333333
```

Nested loops

- Python programming language supports the use of one loop inside another.

Syntax:

for iterating_var in sequence:

```
for iterating_var in sequence:

 Statements(s)

 statements(s
```

In [20]:
```
for i in range(1,4):
 for j in range(1,4):
     print(i,j)
```

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

*

**

***

In [19]:
```
rows = int(input('enter number of rows'))

for i in range(1,rows+1):
  for j in range(1,i+1):
    print('*',end='')
  print()
```

```
enter number of rows5
*
**
***
****
*****
```


Loop Control Statement

- Break
- Continue
- Pass

Break

- Python supports the break statement to implement middle-exiting control logic. The break statement leads to an immediate exit from a loop.

In [22]:
```
#find the first n term sum if n=6 ,break the loop
sum=0
n=int(input("enter number:"))
'''1 for starting range
n+1 for ending range it is not include
last 1 denote the increment or decrement'''
for i in range(1,n+1,1):
 if i==6:
     break
 sum+=i
 print(i)

print("total sum is: ",sum)
```

```
enter number:5
1
2
3
4
5
total sum is:  15
```

In [23]:
```
#find the first n term sum if n=6 ,break the loop
sum=0
n=int(input("enter number:"))
'''1 for starting range
n+1 for ending range it is not include
last 1 denote the increment or decrement'''
for i in range(1,n+1,1):
 if i==6:
     break
 sum+=i
 print(i)

print("total sum is: ",sum)
```

```
enter number:10
1
2
3
4
5
total sum is:  15
```

Continue

- Python supports the continue statement, which returns the control to the beginning of the current loop. When the continue is found, the loop starts the next iteration without executing the remaining statements in the current iteration. The continue statement is used in both while and for loops.

In [27]:
```
#find the first n term sum if n=6 ,continue the loop sum=0
n=int(input("enter number:"))
'''1 for starting range
n+1 for ending range it is not include
last 1 denote the increment or decrement'''
sum=0
for i in range(1,n+1,1):
 if i==6:
     continue
 sum+=i
 print(i)

print("total sum is: ",sum)

```

```
enter number:10
1
2
3
4
5
7
8
9
10
total sum is:  49
```

In [ ]:
```
#example infinite loop
x=1
while x<10:
 print(x,"hello")
 if x==5:
     continue
 x+=1

```

```
1 hello
2 hello
3 hello
4 hello
5 hello
5 hello
5 hello
5 hello
5 hello
5 hello
5 hello
5 hello
5 hello
5 hello
5 hello
5 hello
5 hello
5 hello
5 hello
5 hello
5 hello
```

Pass

- Python supports which is a null statement. The interpreter differentiates a comment and passes by completely ignoring a comment and supporting However, nothing happens when it is executed, and it results in no operation (NOP). Pass statements are used when your code will eventually execute but has not been drafted yet, that is, in stubs. The usage of pass statements for stubs is an excellent example within Python.

In [1]:
```
x=1
while x<10:
 print(x,"hello")
 if x==5:
     pass
 x+=1
```

```
1 hello
2 hello
3 hello
4 hello
5 hello
6 hello
7 hello
8 hello
9 hello
```

Loop else

- Python supports the inclusion of an else statement in a loop statement.

- If the else statement is included in a for loop in Python when the loop has exhausted iterating the list, the else statement is executed then.
- If the else statement is included in a while loop, it is executed when the condition becomes false.

In [2]:
```
x=1
while x<10:
 print(x,"hello")
 x+=1
else:
 print("bye")
```

```
1 hello
2 hello
3 hello
4 hello
5 hello
6 hello
7 hello
8 hello
9 hello
bye
```

In [4]:
```
x=1
while x<10:
 print(x,"hello")
 if x==5:
     break
 x+=1
else:
 print("bye")
```

```
1 hello
2 hello
3 hello
4 hello
5 hello
```

In [5]:
```
for i in range(1,10,5):
    print(i)
else:
    print("vishal")
```

```
1
6
vishal
```

In [1]:
```
for i in range(1,10):
    pass
else:
    print(i)
```

```
9
```

In [2]:
```
for i in range(10):
    i+=5
else:
    print(i)
```

```
14
```

In [3]:
```
a=1
while a<5:
    a+=1
else:
    print(a)
```

```
5
```

In [4]:
```
for i in range(1,11,3):
    print(i)
```

```
1
4
7
10
```

The provided Python code is a for loop that iterates through a range of numbers. It starts at 1, increments by 3 in each step, and continues until it reaches or exceeds 10. During each iteration, the current value of i is printed. As a result, the code will output the numbers 1, 4, 7, and 10, since these are the values i takes on during the loop.

In [7]:
```
for i in range(11,0,-1):
    print(i)
```

```
11
10
9
8
7
6
5
4
3
2
1
```

In [8]:
```
for i in range(11,0,-3):
    print(i)
```

```
11
8
5
2
```

In [ ]:
```

```