Python provides special constructs to control the execution of one or more statements depending on a condition. Such constructs are called as `control statements` or `control-flow statements`.

The control-flow statements are of three types:

`Selection Statement` - is a statement whose execution results in a choice being made as to which of two or more paths should be followed.

- if construct
- if-else construct
- if-elif-else construct
- Nested if-elif-else construct

`Iterative Statement` - is a statement which executes a set of statements repeatedly depending on a condition.

- while loop
- for loop
- else clause on loop statements

`Control flow Statement` - is a statement which transfers control-flow to some other section of the program based on a condition.

- break statement
- continue statement
- pass statement

Select all the correct statements given below.

- [x] Control-flow statements are those which can control the execution flow.
- [ ] Variable declared inside a block can also be accessed outside the block.
- [x] Iterative statements can repeat the execution of statements depending on a condition.
- [ ] Selection statements can be used to execute other statements repetitively.

Close Reset Submit

The general syntax of if statement in **Python** is,

```
If test expression:
    statement(s)
```

- The **if-construct** is a selection statement, the statements within the block are executed only once when the condition evaluates to True, Otherwise, the control goes to the first statement after the if-construct.
- In **Python**, the body (block of statements) of the If statement is indicated by indentation.
- The body starts with indentation and the first unindented line marks the end.
- **Python** interprets **non-zero values** as `True`. None and **0** are interpreted as `False`.

Here is a simple program to illustrate the simple **if-statement**:

```
num = int(input("num: "))
if (num % 3 == 0):
    print("divisible by 3") # Notice the Indentation
print("End of program")
```

Take an integer as input from the console using `input()` function. Write a program to check whether the given integer is divisible by 3 or not, and print the result to the console as shown in the examples.

Sample Input and Output 1:

```
num: 66
divisible by 3
End of program
```

Sample Input and Output 2:

```
num: 52
End of program
```

Correct/complete the code.

lfcon01.py

```
1 #Program to illustrate simple if statement
2 num = int(input("num: "))
3 if(num % 3 == 0):
4     print("divisible by 3")
5 print("End of program")
```

Terminal | Execution Results

Terminal Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

Select the correct output for the below **Python** code?

```
a = 3
b = 3.0
if(a == b):
    print("a and b are equal")
if(a != b):
    print("a and b are not equal")
```

- [ ] Interpreter Error
- [x] a and b are equal.
- [ ] a and b are equal a and b are not equal.
- [ ] a and b are not equal.

Close Reset Submit

The **if-else statement** provides two different paths of execution depending on the result of the condition.

- The body of `if` is executed when the condition associated with the expression is true.
- The body of `else` part is executed when the condition is evaluated to false.
- Indentation is used to separate both the if and else blocks.

Below is the general syntax for the if-else statement :

```
if(expression):
    body of If
else:
    body of else
```

Write a program to check whether the marks obtained by the student got distinction or not. Take marks as input from the user which of type `int`.

Follow the below conditions while writing the program.

- If `marks > distinction _marks` print the message as **distinction**
- Otherwise print the message **not distinction**

Print the result to the console as shown in the examples.

Sample Input and Output 1:

```
marks: 78
distinction
```

Sample Input and Output 2:

```
marks: 55
not distinction
```

NOTE: The distinction_marks = **75** is already defined in the program.

Correct/complete the code. The code highlighted in ■ is non-editable.

Ifelse01.py

```
1 distinction_marks = 75
2
3 # write your code here
4 marks = int(input("marks: "))
5 if(marks> distinction_marks):
6     print("distinction")
7 else:
8     print("not distinction")
```

Terminal | Execution Results

Terminal Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

jionet2.jio.in | How to create 3 boxes in the same div | RESTAURANT | Student Dashboard | CodeTantra Edu

lovelyprofessionaluniversity.codetantra.com/secure/topic-details1.jsp?cid=63049a7c73c69806ec5ea738&tid=63049a8473c69806ec5ea7ca&bd=AMT...

Unit 2 - Lesson 2 / If-else Statement / Q2

Q2

100%

Problem Solving question

Take an integer as input from the console using `input()` function. Write a program to check whether the input amount is greater or less than the minimum balance.

Follow the instructions while writing the program and print the output as shown in the example.

- Assume minimum balance is **1000**
- If `input >= 1000` print Sufficient balance
- Otherwise the message should print on the console as Balance is Low

Sample Input and Output 1:

```
balance: 500
low
```

Sample Input and Output 2:

```
balance: 2000
sufficient
```

Correct/complete the code.

Ifelse02.py

```
1  b = int(input("balance: "))
2  mini = 1000
3  if(b>=mini):
4      print("sufficient")
5  else:
6      print("low")
```

Terminal | Execution Results

Terminal Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/algorithms && cd ct-python-work/algorithms
code@tantra:$
```

16°C Haze | Search | ENG IN | 10:05 PM 11/19/2022

Unit 2 - Lesson 2 / If-else Statement / Q3

Q3

100%

Write a program for Income Tax Calculator

Write a program to calculate the income tax as follows

Step 1: get all deductions (80c, 80cc, HRA, Medical)
Step 2: Add all the deductions to the standard deduction of 150000
Step 3: Get Gross Income
Step 4: Get taxable income (as Gross Income - Deductions)
Step 5: Calculate Income Tax based on taxable income as follows

```
if (Tax_Income > 0):
    if (Gross_Income <= 500000):
        Income_Tax = (Tax_Income * .1)
    if (Gross_Income <= 1000000) and (Gross_Income > 500000):
        Income_Tax = 25000 + ((Gross_Income - 500000)*.2)
    if (Gross_Income > 1000000):
        Income_Tax = 75000 + ((Gross_Income - 1000000) *.3)
    print("Gross Income is " , Gross_Income)
    print("Total Deductions = " , Ded_tot)
    print("Income Tax = " , Income_Tax)
else :
    print("Hurray..No Income Tax")
```

Sample Input and Output 1:

```
deduction under 80c: 50000
deduction under 80cc: 5000
deduction under HRA: 15000
deduction under Medical: 10000
gross income: 250000
gross income: 250000
total deductions: 230000
income tax: 2000.0
```

Sample Input and Output 2:

```
deduction under 80c: 5000
```

Correct/complete the code.

IncomeTaxCal.py

```
# Deductions
Ded_std = 150000
# Request Inputs
Ded_80c = int(input("deduction under 80c: "))
Ded_80cc = int(input("deduction under 80cc: "))
Ded_hra = int(input("deduction under HRA: "))
Ded_med = int(input("deduction under Medical: "))
Gross_Income = int(input("gross income: "))
Ded_tot = (Ded_std + Ded_80c + Ded_80cc + Ded_hra + Ded_med)
Tax_Income = Gross_Income - Ded_tot
if(Tax_Income > 0):
    if(Gross_Income <= 500000):
        Income_Tax = (Tax_Income * 0.1)
    if(Gross_Income <= 1000000 and Gross_Income > 500000):
        Income_Tax = 25000 + ((Gross_Income - 500000) * 0.2)
    if(Gross_Income > 1000000):
        Income_Tax = 75000+ ((Gross_Income - 1000000)*0.3)
    print("gross income",Gross_Income)
    print("total deductions =",Ded_tot)
    print("income tax =",Income_Tax)
else :
    print ("hurray..no income tax")
```

Terminal | Execution Results

Terminal Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/algorithms && cd ct-python-work/algorithms
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

The `if-elif-else` construct extends the if-else construct by allowing to chain multiple if constructs as shown below:

```
if test expression:
    body of if
elif test expression:
     body of elif
elif test expression:
     body of elif
    ...
elif test expression:
     body of elif
else:
     body of else
```

- The `if elif else` construct is used when we have multiple mutually exclusive expressions.
- If the condition for it is false, then the condition for the next elif is evaluated, and so on up to the next elif.
- If all the conditions are false, then the body of else is executed.
- Only one block among `if elif else` blocks is executed based on the condition.
- The if block can have only one else block, but it can have multiple elif blocks.
- Indentation is used for each of the if-elif-else blocks

Take character as input from the console using `input()` function. Write a program to check whether the given input is a `character` or a `digit`, if the input is `0` exit the program, otherwise print the result to the console as shown in the examples.

Sample Input and Output 1:

```
'0' for exit.
ch: 7
digit
```

Sample Input and Output 2:

Correct/complete the code. The code highlighted in ■ is non-editable.

IfelifElse01.py

```
1  print("'0' for exit.")
2
3  # take th input from the user
4  ch = (input("ch: "))
5  if ch == '0':
6      exit()
7  elif (ch >= "0" and ch <= "9"):
8      print("digit")
9  elif (ch >= "a" and ch <= "z") or (ch >= "A" and ch <= "Z"):
10     print("alphabet")
11 else:
12     print("neither alphabet nor digit")
13
14
15 # write your logic here to find the given input is character or digit
```

Terminal | Execution Results

Terminal Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

Unit 2 - Lesson 3 / If-elif-else Statement / Q2

Q2

100%

Write a program to check whether the given number is positive or not.

Take an integer `num` as input from the console using `input()` function. Write a program to check the given `num` is a `positive` or a `negative` one, print the result to the console as shown in the examples.

Sample Input and Output 1:

```
num: 02
positive
```

Sample Input and Output 2:

```
num: -52142
negative
```

Sample Input and Output 3:

```
num: 0
zero
```

Correct/complete the code.

Number.py

```
num = int(input("num: "))
if(num > 0):
    print("positive")
elif(num < 0):
    print("negative")
else:
    print("zero")
```

Terminal | Execution Results

Terminal Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

Unit 2 - Lesson 3 / If-elif-else Statement / Q3

Q3

100%

Program to check leap year

Take an integer `year` as input from the console using `input()` function. Write a program to check the given `year` is a `leap year or not`, print the result to the console as shown in the examples.

Sample Input and Output 1:

```
year: 2004
leap year
```

Sample Input and Output 2:

```
year: 1700
not leap year
```

Correct/complete the code.

Leapyear.py

```
year = int(input("year: "))
if(year % 400 == 0)and(year % 100 == 0):
    print("leap year")
elif(year % 4 == 0)and ( year % 100 != 0):
    print("leap year")
else:
    print("not leap year")
```

Terminal | Execution Results

Terminal — Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/algorithms && cd ct-python-work/algorithms
code@tantra:$
```

Understanding While loop

Syntax of **while** loop in **Python**:

```
while test expression:
    body of while
```

A `while` statement is used to execute some block of code **repeatedly as long as a** `condition` **evaluates to** `True`.

- In the While loop, the value of the expression is evaluated first.
- The body of the loop is entered only when the expression evaluates to true.
- After one iteration, the expression is checked again.
- This process continues until the expression evaluates to `False`.
- The body of the loop is identified using the **indentation** with the expression condition as the first statement.

Take an integer `num` as input from the user. Write a code to find `sum of all even numbers` up to `num` using while construct, print the result to the console as shown in the example.

Sample Input and Output:

```
num: 100
sum: 2550
```

Hint:

sum = 2 + 4 + 8 + ... + n (if n is even)

sum = 2 + 4 + 8 + ... + n - 1 (if n is odd)

Hints

Click on the Live Demo button to know about the while loop in **Python**.

Correct/complete the code.

SumN.py

```
num = int(input("num: "))
i = 2
sum = 0
while(i <= num):
    if(i % 2 == 0):
        sum = sum + i
    i += 1
print("sum:",sum)
```

Terminal | Execution Results

```
Terminal                                   Theme
Welcome to CodeTantra's Live Linux Console!
code@tantra:$
```

While loop with else:

- We can have an optional **else** block with **while** loop as well.
- The **else** part is executed if the condition in the **while** loop evaluates to `False`.
- The while loop can be terminated with a **break** statement. In such a case, the else part is ignored.
- Hence, a **while** loop's else part runs if no break occurs and the condition is `False`.

```
while condition:
    statement_1
    ...
    statement_n
else:
    statement_1
    ...
    statement_n
```

Take an integer `num` as input from the console using `input()` function. Write a program to find the **sum of all integers** between **0** and `num`, both inclusive. Print the result to the console as shown in the examples.

Sample Input and Output 1:

```
num: 250
sum: 31375
```

Sample Input and Output 2:

```
num: -660
sum: -218130
```

Correct/complete the code.

SumN02.py

```
# Python program to find the sum of integers between 0 and n where n is provided by user
n = int(input("num: "))
i = 0
sum = 0
while(n != 0):
    if(n < 0):
        sum = sum + n
        n += 1
        if(n == 0):
            break
    else:
        sum = sum + n
        n -= 1
        if(n == 0):
            break
print("sum:",sum)
```

Terminal | Execution Results

```
Terminal                                                    Theme
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

Unit 2 - Lesson 4 / While Loop / Q3

Q3

100%

Write a program to find the G.C.D. of two given numbers

The **greatest common divisor** (also known as greatest common factor, highest common divisor, or highest common factor) of a set of numbers is the largest positive integer number that divides all the numbers in the set without remainder. It is the biggest multiple of all numbers in the set.

Take two integers `x` and `y` as input from the console using `input()` function. Write a program to find the `GCD` of given inputs `x` and `y`, print the result to the console as shown in the example.

Constraints:

- If one of the inputs is **zero** print the message to the console as shown in the **Sample Input and Output 2**.

Sample Input and Output 1:

```
x: 50
y: 50
gcd: 50
```

Sample Input and Output 2:

```
x: 20
y: 0
value must be non zero
```

Correct/complete the code.

SumofN.py

```
1  x = int(input("x: "))
2  y = int(input("y: "))
3  if( x == 0 or y == 0):
4      print("value must be non zero")
5      quit()
6  i = 1
7  gcd = 0
8  while(i <= x and y):
9      if(x % i == 0 and y % i == 0):
10         gcd = i
11     i += 1
12 print("gcd:",gcd)
```

Terminal | Execution Results

Terminal — Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

Write a program to Calculate Fibonacci numbers less than a given number and calculates the sum of all alternate numbers (even numbered) in the generated list

Take an integer `x` as input from the console using `input()` function. Calculate `Fibonacci series` one number less than the given input `x`, and also calculate the sum of all alternate numbers (Even-numbered) in the resultant list. Print the result to the console as shown in the example.

Sample Input and Output 1:

```
k: 25
0
1
1
2
3
5
8
13
21
sum: 33
```

Explanation: Starting index of first digit is 0, So the sum of even-numbered digits is 0 + 1 + 3 + 8 + 21 = 33

Correct/complete the code.

Fibonacci.py

```
# Write your code here
x = int(input("k: "))
a = 0
b = 1
c = 0
i = 0
l = []
sum = 0
while(a < x):
    print(a)
    l.append(a)
    c = a + b
    a = b
    b = c
while(i < (len(l) )):
    sum = sum + l[i]
    i += 2
print("sum:",sum)
```

Terminal | Execution Results

Terminal — Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

jionet2.jio.in | How to create 3 boxes in the same div | RESTAURANT | Student Dashboard | CodeTantra Edu

lovelyprofessionaluniversity.codetantra.com/secure/topic-details1.jsp?cid=63049a7c73c69806ec5ea738&tid=63049a8573c69806ec5ea7e6&bd=AM...

Unit 2 - Lesson 4 / While Loop / Q5

Q5

100%

Write the code.

Write a program to print all the numbers less than a given number whether they are even or odd using a while loop
Read an integer `x` as input from the console using `input()` function.
For each number less than `x`, print the result to the console whether the number is even or odd as shown in sample output below.

Sample Test Cases

Test Case 1:

Expected Output:

```
k: 5
0 even number
1 odd number
2 even number
3 odd number
4 even number
```

Test Case 2:

Expected Output:

```
k: 15
0 even number
1 odd number
2 even number
3 odd number
4 even number
5 odd number
```

Correct/complete the code.

OddandEven.py

```
x =int(input("k: "))
i = 0
while(i < x):
    if(i % 2 == 0):
        print(i,"even number")
    else:
        print(i,"odd number")
    i += 1
```

Terminal | Execution Results

Terminal — Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

16°C Haze | Search | ENG IN | 10:07 PM 11/19/2022

Write a program to take `capital` and `state` as input from the user, and create a Dictionary with these inputs. The program should terminate once a user enters `end`.

Print the result as shown in the example.

Sample Input and Output:

```
state or 'end' to quit: Andra Pradesh
capital: Amaravathi
state: end
[('Andra Pradesh', 'Amaravathi')]
```

Note:Whenever you print the dictionary (as a whole), please use sorted(dict.items()) so that the items get printed in sorted order (not in random order), otherwise the test cases will fail. The return type of sorted function will be tuples (key,value) rather than key, value pairs separated by `:`

Correct/complete the code. The code highlighted in ■ is non-editable.

StateCap.py

```
2  state = input("state or 'end' to quit: ")
3  if(state == "end"):
4      print(sorted(st2cap.items()))
5      exit()
6  capital = input("capital: ")
7  s = []
8  c = []
9  i = 0
10 s.append(state)
11 c.append(capital)
12 # write your logic using while loop
13 while(True):
14     state = input("state: ")
15     if(state == "end"):
16         break
17     capital = input("capital: ")
18     s.append(state)
19     c.append(capital)
20 # take inputs capital anad state from the user and store it in a dictionary till the user enters state as
21 while(i< len(s)):
22     st2cap[s[i]] = c[i]
23     i +=1
24 print(sorted(st2cap.items()))
```

Terminal | Execution Results

Terminal — Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

A for-loop is used to iterate over a range of values using a loop counter, which is a variable taking a range of values in some orderly sequence (e.g., starting at **0** and ending at **10** in increments of **1**).

The value stored in a loop counter is changed with each iteration of the loop, providing a unique value for each individual iteration. The loop counter is used to decide when to terminate the loop.

A for-loop construct can be termed as an entry controlled loop.

For loop Syntax:

```
for val in sequence:
    Body of for
```

- Here **val** is the variable that takes the value of the item in the sequence for each iteration.
- The Loop continues until the last item in the sequence is reached.
- The body of the for loop is separated from the rest of the code using indentation.

Let us consider a simple example:

```
items = "Python"
index = 0
for item in items:
    print(index, item)
    index += 1
```

Here "**Python**" is a string, we can iterate a string using for loop.

we take '**index**' variable to print the index value of each character.

The output for the above for loop:

```
(0, 'P')
(1, 'y')
(2, 't')
(3, 'h')
```

Correct/complete the code. The code highlighted in ■ is non-editable.

Forloop01.py

```
1  #Program to illustrate simple for loop
2  numbers = [1, 10, 20, 30, 40, 50]
3  sum = 0
4  #  Find sum of all the numbers using for loop
5  for i in numbers:
6      sum = sum + i
7
8  print ("The sum of numbers is", sum ) # print sum here
9
10
11 colors = ['red', 'orange', 'green', 'yellow', 'white', 'violet']
12
13 # Similarly iterate over the given colors and print the colors
14 for i in colors:
15     print(i)
```

Terminal | Execution Results

Terminal Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

The **range()** function :

We can generate a sequence of numbers using **range()** function.
We can use the **range()** function in for loop to iterate through a sequence of numbers.
It can be combined with the **len()** function to iterate through a sequence using indexing.
Syntax:
1. **range(stop) :**
- Returns a sequence of numbers starting from 0 to (stop - 1)
- Returns an empty sequence if stop is negative or 0.
- range(6) will generate numbers from 0 to 5 (up to 6 not including 6).

Let us consider a simple example:

```
for i in range(1, 6):
    print(i)
```

Observe the following output:

```
1
2
3
4
5
```

range(start, stop[, step]) :
The return value is calculated by the following formula with the given constraints:
r[n] = start + step * n (for both positive and negative step)
where, n >= 0 and r[n] < stop (for positive step)
where, n >= 0 and r[n] > stop (for negative step)
- **(If no step)** `Step defaults to 1`. Returns a sequence of numbers starting from start and ending at (stop - 1).
- **(if step is zero)** Raises a ValueError exception
- **(if step is non-zero)** If the value constraint is met, returns a sequence according to the formula
- If it doesn't meet the value constraint, the Empty sequence is returned.
(For example: range(20, 30, -1); because the resulting sequence will never end)

Correct/complete the code. The code highlighted in ■ is non-editable.

PrintRange.py

```
1  #Program to illustrate simple range function
2
3  # take the input num from the user
4  num = int(input("num: "))
5  for i in range(1, num + 1, 2):
6      print(i)
7
8  # print i
```

Terminal | Execution Results

```
Terminal                                                    Theme
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

- A `for loop` can have an optional `else` block as well.
- The **else** part is executed if the items in the sequence used in **for-loop** exhausts.
- A **break** statement can be used to stop a **for-loop**. In such case, the **else** part is ignored.
- Hence, a for-loop's **else** part runs if no **break** occurs.

A common construct to search for an item is to use a **for-loop** which can terminate in two scenarios,

- If the item is found and a break is encountered, exit out of the for-loop.
- If the item is not found, then the loop completes.

So, in order to find out how the loop exits, the `else` clause is useful.

Basic structure of a for-else construct

```
for i in range(1, 10):
    print(i)
else:  # Executed because no break in for
    print("No Break")
```

In the above example, it prints the numbers from **1** to **9** and then else part is executed, and prints No Break.

Let us consider another example

```
for i in range(1, 10):
    print(i)
    break
else:  # else part is  not executed, because there is break statement in the for loop
    print("No Break")
```

In the above example the output is **1** only.

Fill in the missing code in the program below to print a multiplication table up to a maximum of 20 rows. Even if the number of rows asked by the user is more than 20, we should limit up to 20 rows and print an error message as shown in the **Sample Input Output 2**.

Follow the below constraints while giving the inputs:

- x is a positive integer

Correct/complete the code. The code highlighted in ■ is non-editable.

Multiplication.py

```
# Multiplication table
x = int(input("x: "))
y = int(input("y: "))

# Fill in the missing code below to print a multiplication table for x upto y rows.
# If y is more than 20, print the relevant message as per instructions and limit the number of rows to 20
for i in range(1,y+1):
    if(i<=20):
        print("{0} * {1} =".format(x,i),x*i)

else:
    if y>20:
        print("rows is limited to 20")
```

Terminal | Execution Results

```
Terminal  Theme
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

- `Nested for loop` is a concept where we can define one or more `for` loops one inside another.
- The **inner loop** is repeated for each iteration of the **outer loop**.
- According to the documentation you can nest upto `20` blocks one inside another in all control structure's including loops.

Syntax of nested `for` loops is:

```
for i in range(1, 5):              #line 1
    for j in range(0, i):          #line 2
        print('*', end = ' ')    #line 3
    print()                      #line 4
```

1. In the above example, In line 1 the outer loop will run for **i = 1** to **4**.
2. And in line 2 for each value of `i` using `j` variable the inner loop will run for **j = 0** to **i - 1**.
3. In line 3 The **print('*', end = ' ')** statement will print '*' `i` times in the inner loop (In single line with space as a separator).
4. In line 4 the **print()** statement executed when inner loop terminates, and it move's the cursor to the newline.

The output of above given code is:

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

Now your task is to take `n` as a input from the user and print the above pattern upto the given limit `n`.

Sample Input Output:

```
n: 5
*
* *
* * *
* * * *
```

Correct/complete the code. The code highlighted in ■ is non-editable.

nesting.py

```
1  # Write your code here
2  n = int(input("n: "))
3  for i in range(1,n+1):
4      for j in range(0,i):
5          print('*',end=' ')
6      print()
```

Terminal | Execution Results

Terminal — Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

jionet2.jio.in | How to create 3 boxes in the same div | RESTAURANT | Student Dashboard | CodeTantra Edu

lovelyprofessionaluniversity.codetantra.com/secure/topic-details1.jsp?cid=63049a7c73c69806ec5ea738&tid=63049a8573c69806ec5ea7e8&bd=AM...

Unit 2 - Lesson 5 / for Loop / Q5

Q5

100%

Write a program for Matrix Transposition using Nested For loops

Write a program to find the `transpose` of the given matrix in the below program

Sample Input and Output:

```
Matrix before transposition: [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
Matrix after transposition: [[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

Correct/complete the code. The code highlighted in ■ is non-editable.

MatrixTranspose.py

```
1  matrix = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
2  a = []
3  # find the transpose of the matrix and print the result as shown in the example above
4  print("matrix:",matrix)
5  for i in range(0,len(matrix[0])):
6      col = []
7      for j in range(0,len(matrix)):
8          col.append(matrix[j][i])
9      a.append(col)
10 print("transposed:",a)
```

Terminal | Execution Results

Terminal Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

16°C Haze | Search | ENG IN | 10:09 PM 11/19/2022

jionet2.jio.in | How to create 3 boxes in the same div | RESTAURANT | Student Dashboard | CodeTantra Edu

lovelyprofessionaluniversity.codetantra.com/secure/topic-details1.jsp?cid=63049a7c73c69806ec5ea738&tid=63049a8573c69806ec5ea7e8&bd=AM...

Unit 2 - Lesson 5 / for Loop / Q6

Q6

100%

Write a program to check a given number is perfect number

Take an integer `n` as input from the console using `input()` function. Write a program to check whether the given input `n` is a perfect number or not, and print the result to the console as shown in the examples.

Sample Input and Output 1:

```
n: 6
factors: [1, 2, 3]
perfect number
```

Sample Input and Output 2:

```
n: 32
factors: [1, 2, 4, 8, 16]
not perfect number
```

HInt: A number is called a perfect number if the sum of all its factors (exduding itself) is equal to the number

Correct/complete the code.

Perfectnum.py

```
n = int(input("n: "))
f =[]
sum = 0
for i in range(1,n):
    if(n%i == 0):
        f.append(i)
        sum = sum + i
print("factors:",f)
if(sum == n):
    print("perfect number")
else:
    print("not perfect number")
```

Terminal | Execution Results

Terminal Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

16°C Haze | Search | ENG IN | 10:09 PM 11/19/2022

jionet2.jio.in | How to create 3 boxes in the same div | RESTAURANT | Student Dashboard | CodeTantra Edu

lovelyprofessionaluniversity.codetantra.com/secure/topic-details1.jsp?cid=63049a7c73c69806ec5ea738&tid=63049a8573c69806ec5ea7e8&bd=AM...

Unit 2 - Lesson 5 / for Loop / Q7

Q7

100%

Write a program to print the value of Pi to 25 decimal places

Take an integer `n` from the user using `input()` function. Write a program to print the the value of `Pi` up to `n` decimals. Print the result as shown in the example.

Sample Input and Output:

```
n: 5
3.1
3.14
3.142
3.1416
3.14159
```

Hints

```
Use the output formatting  print('{:.{}f}'.format(pi, no.of decimals))
    You can get the value of pi from math.pi
```

Correct/complete the code. The code highlighted in ■ is non-editable.

Multable10.py

```python
#Program to print value of pi 1 to 25 decimals
import math
pi = math.pi

# write your code here
n = int(input("n: "))
for i in range(1,n+1):
    print("{0:.{1}f}".format(pi,i))
```

Terminal | Execution Results

Terminal Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

16°C Haze | Search | ENG IN | 10:09 PM 11/19/2022

jionet2.jio.in | How to create 3 boxes in the same div | RESTAURANT | Student Dashboard | CodeTantra Edu

lovelyprofessionaluniversity.codetantra.com/secure/topic-details1.jsp?cid=63049a7c73c69806ec5ea738&tid=63049a8573c69806ec5ea7e8&bd=AM...

Unit 2 - Lesson 5 / for Loop / Q8

Q8

100%

Write a program to check the given String is valid expression or not.

Program to check the depth and correctness of parenthesis

Write a program that checks for matching parenthesis, and prints an error if they do not match. Print the depth (nesting level) if they match as shown in the examples.

Sample Input and Output 1:

```
str: (())
valid and depth: 2
```

Sample Input and Output 2:

```
str: ()))
not valid and errors: 2
```

Correct/complete the code.

ValidExp.py

```
import math
s = input("str: ")
op = ""
cl = ""
for i in range(0,len(s)):
    if s[i] == '(':
        op = op + '('

    elif s[i] == ')':
        cl = cl + ')'
d = len(op)-len(cl)
if len(op)==len(cl):
    print("valid and depth:",len(op))
else:
    print("not valid and errors:",abs(d))
```

Terminal | Execution Results

Terminal Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

16°C Haze | Search | ENG IN | 10:09 PM 11/19/2022

Loops iterate over a block of code until test expression is **False**, but sometimes we wish to terminate the current iteration or even the whole loop without checking test expression.

Python break statement

The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.

If break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.

Syntax of break

```
break
```

Let us consider a simple example:

```
num = 10
i = 1
while (i <= num):
    if (i % 5 == 0):
        break
    print(i)
    i = i + 1
```

You will notice that once the condition `num % 5 == 0` evaluates to `True` the break statement is executed and it transfers the control out of the loop and further numbers are not printed.

Fill the missing code in the below program to check the `num` is divisible by 5 or not. If the number is divisible by 5 break the program and print the number.

Hints

Click on the Live Demo button to understand the usage of break statement in **Python**.

Correct/complete the code. The code highlighted in ■ is non-editable.

Breakex01.py

```
1  for num in range(1, 10):
2      if(num%5 == 0):
3          break
4      print(num)
5  # write your code here
6
7
8
```

Terminal | Execution Results

Terminal — Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

Write a program to find a given letter is vowel

Take a character input from the user using `input()` function. Write a program to check whether the given input is a vowel or not. Print the result to the console as shown in the examples.

Follow the below constraints while writing the program.

- If the input is an alphabet, then the program should ask the user to enter the input again.
- If the input is a digit other than **9**, the program should print **wrong input**.
- If the input is **9**, the program should exit.

Sample Input and Output 1:

```
vowel, or 9 to quit: a
vowel
vowel, or 9 to quit: w
not vowel
vowel, or 9 to quit: 5
wrong input
vowel, or 9 to quit: 9
```

Sample Input and Output 2:

```
vowel, or 9 to quit: o
vowel
vowel, or 9 to quit: O
vowel
vowel, or 9 to quit: 9
```

Correct/complete the code.

Vowels.py

```
c = input("vowel, or 9 to quit: ")
while True:
    if(c == '9'):
        break
    elif(c in ("a","e","i","o","u","A","E","I","O","U")):
        print("vowel")
    elif(c in ("0","1","2","3","4","5","6","7","8")):
        print("wrong input")
    else:
        print("not vowel")
    c = input("vowel, or 9 to quit: ")
```

Terminal | Execution Results

Terminal — Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

Python continue statement:
The continue statement is used to skip the rest of the code inside a loop of the current iteration. Loop does not terminate but continues with the next iteration.

Syntax of Continue

```
continue
```

Let us consider a simple example:

```
for num in range(1, 10):
    if (num % 2 == 0):
        continue
    print(num)
```

The above code skips printing the even numbers and prints only the odd numbers.

Write the missing code in the below program. Follow the instructions given in the comment lines.

Hints

Click on the Live Demo button to understand the usage of continue statement in **Python**.

Correct/complete the code. The code highlighted in ■ is non-editable.

Continueex01.py

```
1 for num in range(1, 20):
2     if num <= 10:
3         continue
4     print("num:",num)
```

Terminal | Execution Results

Terminal Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

Understanding pass statement

- A `pass` statement in Python has no operation or a null operation statement and when encountered and executed nothing happens.
- It is useful as a placeholder when a statement is required syntactically, but no code needs to be executed.
- It is similar to a comments in Python, however comments are ignored by the interpreter whereas for pass statement it is not. It is still counted as a valid statement.
- A good way to use pass is to hold the place of code that isn't ready or hasn't been written yet. Often it takes the place of loops or functions.
- To use it, type the word **pass** where you would normally insert any other code (like a loop or a function)

Write a program to print all **even numbers** from the list of numbers given below. Ignore all odd numbers using the **pass** statement

```
List <- a list of number
for each number in the list:
    if the number is odd
        pass# do-nothing statement in the form of pass
    else
        print even number
```

Follow the given instructions and write the code in the space given below:

Correct/complete the code. The code highlighted in ■ is non-editable.

Printeven.py

```
#Program to illustrate the pass construct

numbers = [ 1, 2, 4, 3, 6, 5, 7, 10, 9 ]
for i in numbers:
    if i%2 != 0:
        pass
    else:
        print(i)

    #check if the number is odd


            #if odd, then pass ( No operation )

        #print the even numbers
```

Terminal | Execution Results

```
Terminal                                                    Theme
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

Python Program to print Fibonacci series

Take an integer `n` as input from the console using `input()` function. Write a program to calculate the `Fibonacci series` i.e., `0 1 1 2 3 5 8 13 21.....,` up to the given limit `n`, print the result to the console as shown in the example.

Sample Input and Output:

```
n: 5
0
1
1
2
3
5
```

Hint: By definition, the first two numbers in the `Fibonacci` sequence are `0` and `1`, and each subsequent number is the sum of the previous two.

Correct/complete the code.

FibonacciSeries.py

```
1  n = int(input("n: "))
2  a = 0
3  b = 1
4  c = 0
5  while a <= n:
6      print(a)
7      c = a+b
8      a=b
9      b=c
10
```

Terminal | Execution Results

```
Terminal                                                    Theme
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/algorithms && cd ct-python-work/algorithms
code@tantra:$
```

jionet2.jio.in | How to create 3 boxes in the same div | RESTAURANT | Student Dashboard | CodeTantra Edu

lovelyprofessionaluniversity.codetantra.com/secure/topic-details1.jsp?cid=63049a7c73c69806ec5ea738&tid=63049a8473c69806ec5ea7d5&bd=AM...

Unit 2 - Lesson 9 / Practice Programs - Control Statements / Q2

Q2

100%

Write a python program to print prime numbers in a range

Take two integers `x` and `y` as input from the console using `input()` function, consider `x` as lower limt and `y` as upper limit. Write a program to find all the `prime numbers` between `x` and `y`

Sample Input and Output:

```
x: 10
y: 30
11
13
17
19
23
29
```

Hint: A prime number is a positive integer greater than 1 and is divisible by 1 and itself only. A few prime numbers are 2, 3, 5, 7, 11, 13, 17, 19, etc.

Correct/complete the code.

Primeul.py

```
x = int(input("x: "))
y = int(input("y: "))
for i in range(x,y+1):
    for c in range(2,i):
        if(i%c == 0):
            break
    else:
        print(i)
```

Terminal | Execution Results

Terminal Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/algorithms && cd ct-python-work/algorithms
code@tantra:$ cd $HOME && mkdir -p ct-python-work/algorithms && cd ct-python-work/algorithms
code@tantra:$
```

16°C Haze | Search | ENG IN | 10:12 PM 11/19/2022

Write a python program to find if a number is armstrong or not

An **Armstrong Number** is an n digit number that is sum of **nth** power of its digits.

153 = 1 ^ 3 + 5 ^ 3 + 3 ^ 3 = 1 + 125 + 27 is an Armstrong Number

1634 =1 ^ 4 + 6 ^ 4 + 3 ^ 4 + 4 ^ 4 is also an Armstrong Number

Take an integer as input from the console using `input()` function. Write a program to find whether the given integer is an Armstrong number or not, and print the result to the console as shown in the examples.

Sample Input and Output 1:

```
n: 1634
sum of powers: 1634
armstrong number
```

Sample Input and Output 2:

```
n: 123
sum of powers: 36
not armstrong number
```

Hints

All single digit numbers are Armstrong numbers while there are no 2 digit Armstrong numbers.

If you leave the single digit numbers, the least Armstrong number is

153 = 1 ^ 3 + 5 ^ 3 + 3 ^ 3 = 1 + 125 + 27 is an Armstrong Number

1634 =1 ^ 4 + 6 ^ 4 + 3 ^ 4 + 4 ^ 4 is also an Armstrong Number

Correct/complete the code.

ArmstrongNum.py

```
1  n = int(input("n: "))
2  c = len(str(n))
3  sum = 0
4  e =n
5  while n > 0:
6      d=n%10
7      n = n//10
8      sum = sum +(d**c)
9  print("sum of powers:",sum)
10 if sum == e:
11     print("armstrong number")
12 else:
13     print("not armstrong number")
```

Terminal | Execution Results

```
Terminal                                                    Theme
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/algorithms && cd ct-python-work/algorithms
code@tantra:$ cd $HOME && mkdir -p ct-python-work/algorithms && cd ct-python-work/algorithms
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

Program to find a character is a vowel or consonant

Take character as input from the console using `input()` function. Write a program to check whether the given input is a `vowel` or a `consonant` or a `letter` are not. Print the result to the console as shown in the examples.

Sample Input and Output 1:

```
ch: A
letter and vowel
```

Sample Input and Output 2:

```
ch: P
letter and consonant
```

Sample Input and Output 3:

```
ch: @
not letter
```

Correct/complete the code.

IsAlphaVowel.py

```
1  # Type your program here...
2  c = input("ch: ")
3  import string
4  a = list(string.ascii_lowercase)
5  c = c.lower()
6  a.remove("a")
7  a.remove("e")
8  a.remove("i")
9  a.remove("o")
10 a.remove("u")
11 if c in ("a","e","i","o","u"):
12     print("letter and vowel")
13 elif c in a:
14     print("letter and consonant")
15 else:
16     print("not letter")
```

Terminal | Execution Results

```
Terminal                                                    Theme
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/algorithms && cd ct-python-work/algorithms
code@tantra:$ cd $HOME && mkdir -p ct-python-work/algorithms && cd ct-python-work/algorithms
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

Numbers plays vital role in our day to day life. **Python** provides 3 types of numbers as we discussed in previous sessions.

1. **Integers.**
2. **Floating Point or Real Numbers.**
3. **Complex Numbers.**

Integers - The numbers without decimal point.

For example: `11`, `15`, `8` are integers but **2.0**, **10.2** are not.

Floating point or real numbers - the numbers which have decimal points. For example, `1.6`, `52.3`, `3.6` are float but **12**, **3** are not.

Complex numbers - the numbers which we can not be represented on a number line. A Complex number is in the form `a + bj`, where `a` is the real part and `bj` is the imaginary part.
For example: `7 + 6j` is complex number.

Python uses a special syntax for complex numbers too. A integer or float with trailing `j` is treated as a complex number in **Python**, so `16j`, `1 + 2j` are complex numbers.

Let us discuss a built-in function called **isinstance()** to know the object belongs to respective class type or not.

```
print(isinstance(7, int)) # will print result as follows
True
print(isinstance(7.020, int)) # will print result as follows
False
```

Here **isinstance()** method returns Boolean values either `True` or `False`. **7** is an int value so it returns True. **7.020** is float number not int so returns False.

Correct/complete the code.

IntroNumber1.py

```
# find quotient using '/' and '//' like (a/b) and (a//b)
a = int(input("a: "))
b = int(input("b: "))
print("{0} / {1} = {2}".format(a,b,a/b))
print("{0} // {1} = {2}".format(a,b,a//b))
print("{0} % {1} = {2}".format(a,b,a%b))
```

Terminal | Execution Results

Terminal Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$
```

jionet2.jio.in | How to create 3 boxes in the same div | RESTAURANT | Student Dashboard | CodeTantra Edu

lovelyprofessionaluniversity.codetantra.com/secure/topic-details1.jsp?cid=63049a7c73c69806ec5ea738&tid=63049a8073c69806ec5ea741&bd=AM...

Unit 2 - Lesson 10 / Introduction to Numbers / Q2

Q2

100%

Write a program to addition, subtraction and multiplication and division of two complex numbers.

Take **two complex numbers** as input from the console using `input()` function. For each arithmetic operator(`+`, `-`, `*`, and `/`), print the result to the console, the result of the two inputs after performing these opearations as shown in the example.

Sample Input and Output:

```
c1: 10+3j
c2: 14+3j
a1 + b2 = (24+6j)
a1 - b2 = (-4+0j)
a1 * b2 = (131+72j)
a1 / b2 = (0.7268292682926829+0.05853658536585366j)
```

Correct/complete the code.

ComplexNums.py

```
1  c1 = complex(input("c1: "))
2  c2 = complex(input("c2: "))
3  print("a1 + b2 =",c1+c2)
4  print("a1 - b2 =",c1-c2)
5  print("a1 * b2 =",c1*c2)
6  print("a1 / b2 =",c1/c2)
```

Terminal | Execution Results

Terminal Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

16°C Haze | Search | ENG IN | 10:13 PM 11/19/2022

Python provides below built-in functions to perform number conversion. Sometimes, you need to convert a number explicitly from one type to another to satisfy the requirements of an operator or arguments of functions.

We already discussed these number conversions in type conversions.

1. `int(x)` to convert given **x** value to a plain integer.

> **Note:** A string can be converted into an integer using the int() function.
> For Example :
> `str = "1010"`
> `num = int (str,2)` *# This will convert the str to integer assuming the base as 2. so the value of number would be equal to* `10`.
>
> `str = "1243"`
> `num = int(str,8)` *# This will convert the str to integer assuming the base as 8. so the value of number would be equal to* `675`.
>
> **Note:** If no base is defined, the default value is 10.

2. `float(x)` to convert given **x** value to a floating-point number.

Remember the type of **5.0** is float and type of **5** is int.

3. `complex(x)` to convert given **x** value to a complex number with real part **x** and imaginary part **zero**.

Remember the type of **9 + 0j** is **complex** and type of **9** is **int**.

4. `complex(x, y)` to convert given **x** and **y** values to a complex number with real part **x** and imaginary part **y**. **x** and **y** are numeric expressions.

In the below program a string is defined with a value `0111110`. Write a program to convert the given input to `int`, `float` and `complex` data types and print the result and data type of the result as shown in the example..

Correct/complete the code.

Number2.py

```
1  # Type Conversion Example
2  a = '0111110'
3  b = int(a,2)
4  print("converting to int: {0} ,data type: {1}".format(b,type(b)))
5  c = float(b)
6  print("converting to float: {0} ,data type: {1}".format(c,type(c)))
7  d = complex(b)
8  print("converting to complex: {0} ,data type: {1}".format(d,type(d)))
```

Terminal | Execution Results

```
Terminal                                                    Theme
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

We can convert an integer(decimal, base 10) to base 2 (binary), base 8 (Octal), and base 16 (Hexadecimal) using built-in functions.

The result of this conversion will be a string

Let us consider example:

```
x = 62
print(bin(x)) # will print result as follows
0b111110
```

```
print(oct(x)) # will print result as follows
0o76
```

```
print(hex(x)) # will print result as follows
0x3e
```

Let us write a small program to illustrate these conversions:

Take an integer as input from the console. Convert the given input integer in to **Binary**,**Octal** and **Hexadecimal** forms, print the result as shown in the example.

Sample Input and Output:

```
x: 63
Decimal: 63
Binary: 0b111111
Octal: 0o77
Hexadecimal:0x3f
```

Correct/complete the code.

BaseConv.py

```
1  c = int(input("x: "))
2  print("Decimal: {0} ".format(c))
3  b = bin(c)
4  print("Binary: {0} ".format(b))
5  o = oct(c)
6  print("Octal: {0} ".format(o))
7  h = hex(c)
8  print("Hexadecimal: {0}".format(h))
```

Terminal | Execution Results

Terminal Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

Python has numerous **mathematical functions**. The full list can be found at Mathematical Functions
To perform mathematical functions we have to import **math** module.

The math module contains **mathematical functions** and **mathematical constants**.

Most of the **mathematical functions** are available in math module except min(), max(), abs(), pow(), round().

They are divided into the following categories:

Number-theoretic and representation functions:

floor() , fmod()

ceil(), copysign() , fabs()
factorial()
frexp() ,fsum() , gcd()
isclose() ,isinf(), isnan()
ldf() , modf() ,trunc()

Power and Logarithmic functions:

exp(), expm(), log(),
log1p(), log2(), log10(),
pow(), sqrt()

Trigonometric functions:

Correct/complete the code.

Round.py

```
1  import math
2  num = float(input("num: "))
3  if num - int(num) >=.5:
4      print ("result:",math.ceil(num))
5  else:
6      print("result:",math.trunc(num))
```

Terminal | Execution Results

Terminal Theme

Close Reset Submit

In real world, we need to generate random numbers quite often. For example, to choose a winner in a lottery, to choose a student to ask question etc.

Python provides us random number functions. We use these random numbers in areas like research, games, cryptography, simulation and other applications.

We have to import **random** module to use random functions.

Let us discuss random functions:

1. `choice(seq)` - This function returns a random element from the non-empty sequence. If **seq** is empty, interpreter raises an `IndexError`.

```
import random
seq = "abcdefghijklmnopqrstuvw"
print(random.choice(seq)) # will print result as follows
d
```

2. `shuffle(list)` - This functions returns shuffled list.

```
L1 = [10, 20, 2, 3, 1]
random.shuffle(L1)
print (L1) # will print result as follows
[1, 10, 3, 20, 2]
```

3. `randint(a, b)` - This function returns a random integer between **a** and **b** indusive

```
print(random.randint(1, 5)) # will print result as follows
2
print(random.randint(1, 5)) # will print result as follows
5
```

- [x] **random()** function returns random values between 0.0 and 1.0 only.
- [ ] If we are not specifying **step** in **randrange()**, interpreter raises an Error.
- [ ] using **shuffle()** function we can reshuffle the tuple.

Close Reset Submit

Trigonometry is very useful in real Life.

Python provide us **Trigonometric functions** to perform trigonometric operations.

All the trigonometric functions are available in **math module**.

We have to import math module to use trigonometric functions.

1. sin() - This function returns the sine of given value. The value should be in radians.

```
import math
print(math.sin(90)) # will print result as follows
0.8939966636005579
print(math.sin(1.5708)) # 90 degrees = 1.5708 radians
0.9999999999932537  # Almost 1
```

2. cos() - This function returns the cosine of given value. The value should be in radians.

```
import math
print(math.cos(90)) # will print result as follows
-0.4480736161291701
```

3.tan() - This function returns tangent of given value. The given value should be in radians.

```
print(math.tan(90)) # will print result as follows
-1.995200412208242
```

4.asin() - This function returns the arc sine of given value.

```
print(math.asin(1)) # will print result as follows
1.5707963267948966
```

Correct/complete the code. The code highlighted in ■ is non-editable.

SinCosTan.py

```
from math import *

print("degrees 0 30 45 60 90")
a = radians(0)
b = radians(30)
c = radians(45)
d = radians(60)
e = radians(90)

# write your code here
print("sin {0} {1} {2} {3} {4}".format(sin(a),sin(b),sin(c),sin(d),sin(e)))
print("cos {0} {1} {2} {3} {4}".format(cos(a),cos(b),cos(c),cos(d),cos(e)))
print("tan {0} {1} {2} {3} {4}".format(tan(a),tan(b),tan(c),tan(d),tan(e)))
```

Terminal | Execution Results

Terminal Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

Python provides five mathematical constants in math module which are `math.pi`, `math.e`, `math.tau`, `math.inf` and `math.nan`

1. math.pi - The mathematical constant π = 3.141592..., to available precision.

2. math.e - The mathematical constant e = 2.718281..., to available precision.

3. math.tau -The mathematical constant τ = 6.283185..., to available precision. Tau is a circle constant equal to 2π. It is available in version 3.6

4. math.inf - A floating-point positive infinity. Equivalent to the output of float('inf'). It is available in version 3.5.

5. math.nan - A floating-point "not a number" (NaN) value. Equivalent to the output of float('nan')

```
print(math.pi) # will print result as follows
3.141592653589793
```

pi constant returns value of pi

```
print(math.e) # will print result as follows
2.718281828459045
```

returns an exponential value.

```
print(math.inf) # will print result as follows
inf
```

```
print(math.nan) # will print result as follows
nan
```

Follow the instructions given in the comment lines and write the code in the space provided to get the result as

Correct/complete the code. The code highlighted in ■ is non-editable.

Constants.py

```
1  # Write a program to print Mathematical constants
2  import math
3
4  print("constant\tvalue\tdata type")
5
6  print("pi\t", math.pi, "\t",type(math.pi)) # find type of math.pi using type function
7
8  print("e\t", math.e, "\t",type(math.e))# find type of math.e
9
10 print("inf\t", "{:17}".format(math.inf), "\t",type(math.inf)) # find type of math.inf
11
12 print("NaN\t", "{:17}".format(math.nan), "\t",type(math.nan)) # find type of math.nan
```

Terminal | Execution Results

Terminal — Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```

Program to print values of mathematical constants

Write a program to print values of all mathematical constants (`pi` , `e` , `inf` and `NaN`). For each mathematical constant, print the result to the console as shown in the example.

Sample Input and Output:

```
Constant      Value          data type
Pi        3.141592653589793 <class 'float'>
e         2.718281828459045 <class 'float'>
inf             inf         <class 'float'>
NaN             nan         <class 'float'>
```

Correct/complete the code.

Constants.py

```
1  from math import *
2  print("constant\tvalue\tdata type")
3  print("pi\t",pi,"\t",type(pi))
4  print("e\t",e,"\t",type(e))
5  print("inf\t","{:17}".format(inf),"\t",type(inf))
6  print("NaN\t","{:17}".format(nan),"\t",type(nan))
```

Terminal | Execution Results

Terminal — Theme

```
Welcome to CodeTantra's Live Linux Console!
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$ cd $HOME && mkdir -p ct-python-work/introduction && cd ct-python-work/introduction
code@tantra:$
```