Core of Python Programming

Day8_CorePy.md

by Nathan Hailu

Recalling

LAST TIME TOPIC

Topics

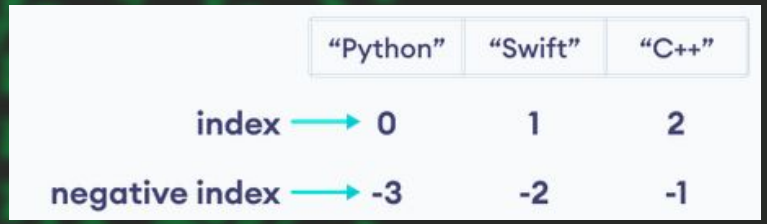
- indexing and slicing{start,stop,step}
- input handling{2 methods}
- Operations
- indentation
- if else
- Errors (Exceptions)
- Error handling
- loops

Indexing

- On lists, we have seen about index numbers.

```
languages = ["Python", "Swift", "C++"]
```

- Negative indexing




```
languages = ["Python", "Swift", "C++"]

# access item at index 0
print(languages[-1])   # C++

# access item at index 2
print(languages[-3])   # Python
```

Note: The list index always starts with **0**. Hence, the first element of a list is present at index **0**, not **1**.

Cont...

- Calling texts by indexes also works for strings & tuples

```
name = ('hello',23,22)
print(name[2])

# Output: 22
```

```
name = 'Nathan'
print(name[2])

# Output: t
```

```
name = 'Nathan'
print(name[-2])

# Output: a
```

?

Slicing

- In Python it is possible to access a section of items from the list using the slicing operator ‘:’ , not just a single item.
- Syntax:
  - Mylist[start : stop : step]
- Slicing is indexing syntax that extracts a portion from a list. If a is a list, then a[m:n] returns the portion of a:
  - Starting with postion m
  - Up to but not including n
  - Negative indexing can also be used
- It is more applicable for strings.

```
name = 'No 1 is Nathan'
print(name[8:14:1])


# Output: Nathan
```

- Python uses default step as 1, sometimes no need to tell/put it
- Also default stopping index is the final, still no need to tell for this kinda purpose

Cont...

There are more writing features.

```
name = ['ethiopia','banana','china','apple']
print(name[::])

# Output: ['ethiopia','banana','china','apple']
```

```
name = 'No 1 is Nathan'
print(name[8:14])

# Output: Nathan
```

```
name = 'No 1 is Nathan'
print(name[8:])

# Output: Nathan
```

```
name = 'No 1 is Nathan'
print(name[8:-1])

# Output: Natha
```

```
name = ['ethiopia','banana','china','apple']
print(name[-1:0:-2])

# Output: ['apple', 'banana']
```

```
indexing  ['ethiopia','banana','china','apple']
- Normal      0          1         2        3
- Negative   -4         -3        -2       -1

STEPs ->      1          2         3        4

              -4         -3        -2       -1  <- STEPs
```

QUick QUiZ - 5 sec foreach - guess the

```
print(name[_:_:_])
```

1) output?

```
name = ['ethiopia','banana','china','apple']
print(name[::-1])
# Output: ['apple', 'china', 'banana', 'ethiopia']
```

2) Another method for this?

```
# Output: ['apple', 'banana']
name = ['ethiopia','banana','china','apple']
print(name[::-2])
```

3) Can we have this output?

```
# Output: countries are: ['china', 'ethiopia']
name = ['ethiopia','banana','china','apple']
print(f"countries are: {name[-2::-2]}")
```

User Input handling

- On python there are 2 types of inputs
  - By input function
  - By Arguments

1) By input functions,
   - Syntax: var = input(“Text you like to display: ”)
   - Will accept the input and **stores on variable**

```
name = input("What is your name?\n name: ")
print(f"Hello {name}!")

# Output: What is your name?
#         name: 'Nathan Hailu'
#         Hello Nathan Hailu!
```

Cont...

You can change the input type to int() float() eval() str()....

```
number = input("Enter number: ")
print(type(number))

# Output: <class 'str'>
```

```
number = int(input("Enter number: "))
print(type(number))

# Output: <class 'int'>
```

```
number = eval(input("Enter number: "))
print(type(number))

# Output: <class 'int'>
```

```
number = float(input("Enter number: "))
print(type(number))

# Output: <class 'float'>
```

Cont...

2) Arguments

- This help us to get the input from the command lines
- Shell: python gtst.py arg1 arg2 arg3
- Syntax:

```
import sys
name = sys.argv[1]
print(f"Hello {name}!")
```

```
PS C:\Users\Nathan Hailu\Desktop> python test.py Nathan Hailu
Hello Nathan!
```

Cont...

- Have You seen the output?
  - We entered -> Nathan Hailu
  - Output: Hello Nathan!
    - WHY??

```
PS C:\Users\Nathan Hailu\Desktop> python test.py Nathan Hailu
Hello Nathan!
```

```
Nathan Hailu    =>    "Nathan Hailu" "Geez Tech"
   1     2                    1              2
```

```
PS C:\Users\Nathan Hailu\Desktop> python test.py "Nathan Hailu"
Hello Nathan Hailu!
```

Further...

You can do more inputs

```
1   import sys
2   firstname = sys.argv[1]
3   lastname = sys.argv[2]
4   print(f"Hello {firstname}!, Your Father name is: {lastname}.")
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\Nathan Hailu\Desktop> python test.py Nathan Hailu
Hello Nathan!, Your Father name is: Hailu.
```

You can create variable until n times

```
var = sys.argv[100]
```

Exercise 1 15min

1. A developer made a software that accepts username and passwords. It stores it in list with username,password sequence. Can you help him for sorting the username and passwords, and give him display of “ the usernames are: …. “ and “Passwords are: …”
    a. The Data He Gave you is
        i. users = ['Nathan',2313,'Geez','pass231','Abebe','092313133','Miki',"pl3s34D0n'tH4ckM3"]
2. Write a code that Prints a password of a user, when the user inputs his username.
    a. HINT: change the upper users list to dictionary.

Operators

- Operators are special symbols that perform operations on variables and values. For example,

```
print(5 + 6)   # 11
```

- There are lots of operators type on python:
  a. Arithmetic operators
  b. Assignment Operators
  c. Comparison Operators
  d. Logical Operators
  e. Bitwise Operators
  f. Special Operators

A) Arithmetic Operators

They are a simple maths operations

Inputs have to be in int,eval, float only

Operator	Operation	Example
+	Addition	5 + 2 = 7
-	Subtraction	4 - 2 = 2
*	Multiplication	2 * 3 = 6
/	Division	4 / 2 = 2
%	Modulo	5 % 2 = 1
**	Power	4 ** 2 = 16

```
a = 7
b = 2

# addition
print ('Sum: ', a + b)
# subtraction
print ('Subtraction: ', a - b)
# multiplication
print ('Multiplication: ', a * b)
# division
print ('Division: ', a / b)
# modulo
print ('Modulo: ', a % b)
# a to the power b
print ('Power: ', a ** b)
```

B) Assignment Operators

- Assignment operators are used to assign values to variables
- You do the arithmetic operators 1st , then the equal sign.

Operator	Name	Example
`=`	Assignment Operator	`a = 7`
`+=`	Addition Assignment	`a += 1 # a = a + 1`
`-=`	Subtraction Assignment	`a -= 3 # a = a - 3`
`*=`	Multiplication Assignment	`a *= 4 # a = a * 4`
`/=`	Division Assignment	`a /= 3 # a = a / 3`
`%=`	Remainder Assignment	`a %= 10 # a = a % 10`
`**=`	Exponent Assignment	`a **= 10 # a = a ** 10`

```
# assign 10 to a
a = 10

# assign 5 to b
b = 5

# assign the sum of a and b to a
a += b      # a = a + b

print(a)

# Output: 15
```

C) Comparison operators

- Used to compare to variables and return boolean result
- Boolean means either TRUE or FALSE

Operator	Meaning	Example
==	Is Equal To	3 == 5 gives us **False**
!=	Not Equal To	3 != 5 gives us **True**
>	Greater Than	3 > 5 gives us **False**
<	Less Than	3 < 5 gives us **True**
>=	Greater Than or Equal To	3 >= 5 give us **False**
<=	Less Than or Equal To	3 <= 5 gives us **True**

```
a = 5
b = 2

# equal to operator
print('a == b =', a == b) # False
# not equal to operator
print('a != b =', a != b) # True
# greater than operator
print('a > b =', a > b) # True
# less than operator
print('a < b =', a < b) # False
# greater than or equal to operator
print('a >= b =', a >= b) # True
# less than or equal to operator
print('a <= b =', a <= b) # False
```

D) Logical Operators

- They are used to check if an expression is TRUE or FALSE
- They use Truth tables to compare.

Operator	Example	Meaning
`and`	a **and** b	**Logical AND:** `True` only if both the operands are `True`
`or`	a **or** b	**Logical OR:** `True` if at least one of the operands is `True`
`not`	**not** a	**Logical NOT:** `True` if the operand is `False` and vice-versa.

Truth table for and / እና (&&)

- **Only True and True is True**

Truth table for and		
A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

Truth table for or / ወይም (||)

- **Only False and False is False**

Truth table for or		
A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

Truth table for not / አይደለም

- It is just opposite.

Truth tabel for not	
A	not A
True	False
False	True

Quick Quiz

```
# logical AND
print(True and True)      # True
print(True and False)     # False

# logical OR
print(True or False)  # True


# logical NOT
print(not True)  # False
```

Bitwise Operators

```
print("3 in binary is: ",bin(3))
print("11 in decimal is: ",int('11',2))

# Output: 3 in binary is: 0b11
#         11 in decimal is: 3
```

- Computers Work with Binarys
- On our computer everything have a binary value. (also called bit)
- On python there is a keyword called bin(YourDecimal) this helps to Show you the binary value of YourDecimal
- True have a value of 1
- False have a value of 0
- Bitwise Operators Used to do maths(Logical operations) on The binary value of The expression.
- There are
  - Compliment (Not) (~)
  - And (&)
  - Or (|)
  - Xor (^)
  - Left Shift (<<)
  - Right shift (>>)

If you work on CryptoGraphy on hacking this is must!

Complement(NOT) (~)

- It will convert the first value to binary and it will reverse each bit then converts to decimal.
- In simple maths, it will add 1 to the number and then makes it negative.

```
~12   =>   -(12+1)   =  -13

~4    =>   -5
```

```
print(~12)

# Output: -13
```

And(&)

- You can add 0 before the binary of any number if it is not 4 digit binary
- bin(7) -> 111 , but we can do 0111 too

```
10&7

10   ->   1010
          &&&&
7    ->   0111
--------------
AND       0010     == 2
```

```
print(10&7)

# Output: 2
```

OR (|)

SAME AS AND but the logic operator will be changed.

```
10&7

10   ->   1010
          ||||
7    ->   0111
--------------
OR        1111     == 15
```

```
print(10|7)

# Output: 15
```

XOR (^)

- It is like and , or but the difference is the logic here is
  - If they are same = 0 1^1 = 0 , 0^0 = 0
  - If they are different = 1 1^0 = 1 , 0^1 = 1

```
10&7

10   ->   1010
          ^^^^
7    ->   0111
--------------
XOR       1101     == 13
```

```
print(10^7)

# Output: 13
```

Left Shift (<<)

- Every Numbers have .0 at the end => 1.0 ,32.0 ....

```
1010.0000
```

```
10<<2 - shifting 2 bits to the left

10   ->   1010.0000
          101000.00
---------------
<<       101000      == 40
```

```
print(10<<2)


# Output: 40
```

Right shift (>>)

```
10>>2 - shifting 2 bits to the Right

10   ->   1010.0000
          10.100000
---------------
>>      10     == 2
```

```
print(10>>2)


# Output: 2
```

indentations

- Are Just a WhiteSpace which python uses for some of its function. If there is no proper indentation error then you are doomed.

```
<?xml version="1.0" encoding="UTF-8"?>
<personnel>
  <person id="Big.Boss">
    <name>
      <family>
         (The end tag is indented at the same level
          as the start tag.)
      </family>
    </name>
    <address>
      <city>
        <zip>
          <code>(The children of an element increase
           the indentation level.)</code>
        </zip>
      </city>
    </address>
```

If-else conditions

- In computer programming, we use the if statement to run a block code only when a certain condition is True.

  For example, assigning grades **(A, B, C)** based on marks obtained by a student.

  1. if the percentage is above **90**, assign grade **A**
  2. if the percentage is above **75**, assign grade **B**
  3. if the percentage is above **65**, assign grade **C**

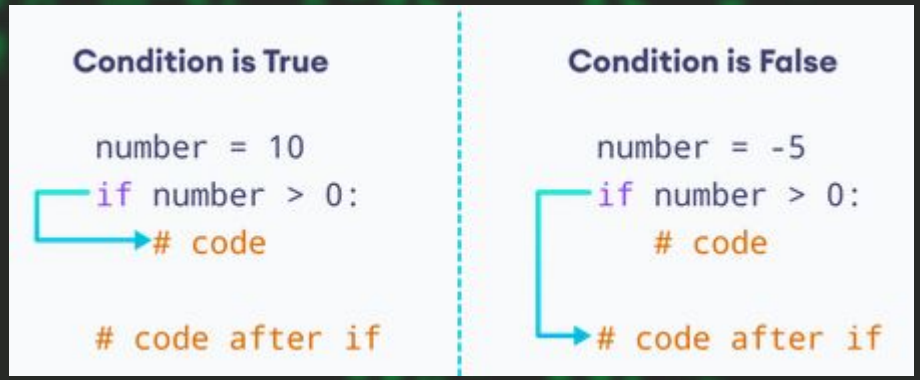
- **In Python, there are three forms of the if...else statement.**
  1. **if statement**
  2. **if...else statement**
  3. **if...elif...else statement**

```
if success() == True:
    celebrate()
while success() == False:
    try_again()
    be_awesome()
```

If statement

- The if statement evaluates condition.
  - If condition is evaluated to True, the code inside the body of if is executed.
  - If condition is evaluated to False, the code inside the body of if is **skipped**.
- Synt
  -

```
if condition:
    # body of if statement
```




```
number = 10
# check if number is greater than 0
if number > 0:
    print('Number is positive.')
print('The if statement is easy')

# Output: Number is positive.
#         The if statement is easy
```

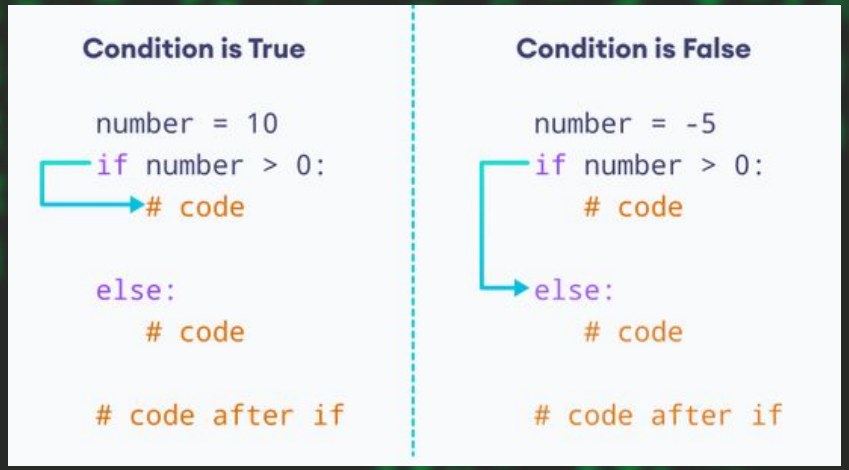
Indentation

If...else statement

An if statement can have an optional else clause.

The syntax of if...else statement is:

```
if condition:
    # block of code if condition is True

else:
    # block of code if condition is False
```



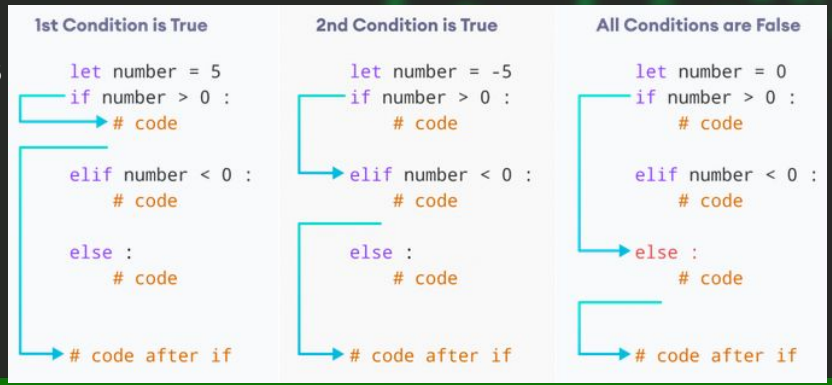

```
number = 10
if number > 0:
    print('Positive number')

else:
    print('Negative number')

print('This statement is always executed')

# Output: Positive number
#         This statement is always executed
```

If...elif...else Statement

1. Here,
2. If **condition1** evaluates to True, **code block 1** is executed.
3. If **condition1** evaluates to False, then **condition2** is evaluated.
4. If **condition2** is True, **code block 2** is executed.
5. If **condition2** is False, **code block 3** is executed.

```
if condition1:
    # code block 1

elif condition2:
    # code block 2

else:
    # code block 3
```

```
number = 0

if number > 0:
    print("Positive number")

elif number == 0:
    print('Zero')
else:
    print('Negative number')

print('This statement is always executed')
```



Nested if statements

- We can also use an if statement inside of an if statement. This is known as a **nested if** statement.
- Here two requirements have to be true to run the body of condition2

```
# outer if statement
if condition1:
    # statement(s)

    # inner if statement
    if condition2:
        # statement(s)
```

```
number = 5

# outer if statement
if (number >= 0):
    # inner if statement
    if number == 0:
      print('Number is 0')

    # inner else statement
    else:
        print('Number is positive')

# outer else statement
else:
    print('Number is negative')

# Output: Number is positive
```

Exercise 2

1) Abel wanted to make a software that accepts a number and checks a number if the number is even it displays “ This number is Even”, if the number is odd “ This number is Odd” if the input is not integer display “Please enter a number only!” else display “nothing inserted!” can you do this program for him?

Logical Errors (Exceptions)

- Errors that occur at runtime (after passing the syntax test) are called **exceptions** or **logical errors**.
- For instance, they occur when we
  - try to call an index that is greater than the list have causes (IndexError)
  - try to divide a number by zero (ZeroDivisionError)
  - When you have error on your syntax (NameError) and so on.
- So specially when errors occur on runtime this causes a huge damage on our program so we have to handle it.

Error handling

- For handling errors we use try…except blocks.

```
try:
    # code that may cause exception
except:
    # code to run when exception occurs
```

```
try:
    numerator = 10
    denominator = 0

    result = numerator/denominator

    print(result)
except:
    print("Error: Denominator cannot be 0.")

# Output: Error: Denominator cannot be 0.
```

```
try:

    even_numbers = [2,4,6,8]
    print(even_numbers[5])

except ZeroDivisionError:
    print("Denominator cannot be 0.")

except IndexError:
    print("Index Out of Bound.")

# Output: Index Out of Bound
```

Python Loops

- In computer programming, loops are used to **repeat a block of code**.
- For example, if we want to show a message **100** times, then we can use a loop. And print(100) It's just a simple example; you can achieve much more with loops.
- There are 2 types of loops in Python:
  - For Loop
  - While Loop

```
while(alive)
{
    eat();
    sleep();
    code();
}
```

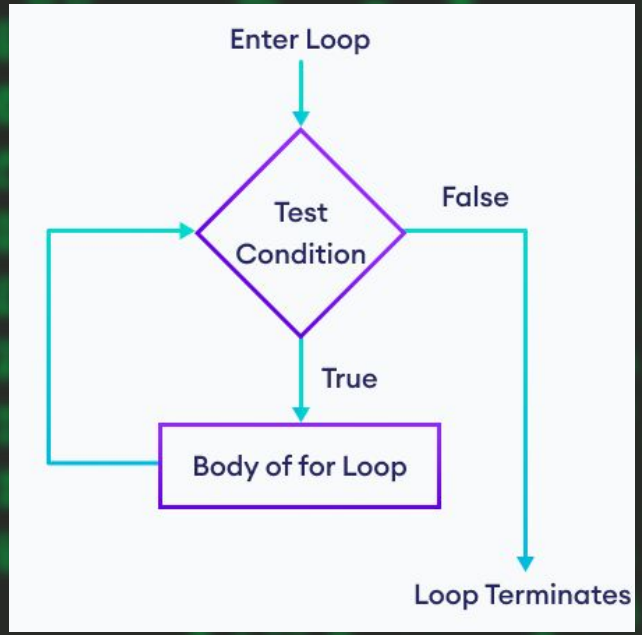
For loop

- In Python, the for loop is used to run a block of code for a certain number of times. It is used to **iterate over any sequences** such as list, tuple, string, etc.

- Syntax:

```
for val in sequence:
    # statement(s)
```

  - Sequence is a list,tuple,string or range.
  - Val is a variable which will hold the iteration from the sequence.




```
languages = ['Swift', 'Python', 'Go', 'JavaScript']

# access items of a list using for loop
for language in languages:
    print(language)
```

```
Swift
Python
Go
JavaScript
```

Cont...

%% Range keyword

A range is series of values between two numeric intervals. range(size)

```
number = range(5)
print(number)

# Output: range(0, 5) <==
```

```
for i in range(5):
    print(i)

# Output: 0
#         1
#         2
#         3
#         4
```

%% len keyword

A len is used to show the length of a sequence may be list,tuple or staffing. len(list)

```
a = [1,2,3,4,5,'hello']

print(len(a))

# Output: 6
```

```
a = [1,2,3,4,5,'hello']

for i in range(len(a)):
    print(i)

# Output: 0
#         1
#         2
#         3
#         4
#         5
```

While loops

- Python while loop is used to run a specific code until a certain condition is met.
- Syntax:

```
while condition:
    # body of while loop
```

```
# program to display numbers from 1 to 5

# initialize the variable
i = 1
n = 5

# while loop from i = 1 to 5
while i <= n:
    print(i)
    i = i + 1
```

Variable	Condition: `i <= n`	Action
`i = 1` `n = 5`	`True`	`1` is printed. `i` is increased to **2**.
`i = 2` `n = 5`	`True`	`2` is printed. `i` is increased to **3**.
`i = 3` `n = 5`	`True`	`3` is printed. `i` is increased to **4**.
`i = 4` `n = 5`	`True`	`4` is printed. `i` is increased to **5**.
`i = 5` `n = 5`	`True`	`5` is printed. `i` is increased to **6**.
`i = 6` `n = 5`	`False`	The loop is terminated.

Cont...

1) You can do infinite loops

```
# infinite while loop
while True:
    # body of the loop
```

Difference between for and while

- The for loop is usually used when the number of iterations is known.
- And while loop is usually used when the number of iterations are unknown. When we have condition.
- Example:
  - If you have a case that wanted to check level of a user and displays " you have passed {n}th level" n is sequence. Until the user level and the class level is equal. What do u do?

```
current_level = 0
final_level = 5
while current_level <= final_level:
    print('You have passed level', current_level)
    current_level += 1
print('Level Ends')
```

- For loops: ends when the iterable is finished.
- While loops: end when the condition is false.

break

Break used to exit from an infinite loop.

```
code = [2313,2314,4325,6546]
errors = 0

while True:
    if errors <= 5:
        user = int(input(f"Enter The captcha correctly {code[0]}:\n>>"))
        if user != int(code[0]):
            print(int(code[0]))
            print(f"trail{errors}: incorrect!, try again")
            errors += 1
        elif user == int(code[0]):
            print("wellDone!")
            break
    else:
        print("try again,next time.")
        break
```

Output

Case:

Write a program that helps to check a code , if the users errors are 5 close the program and display “ try again next time” else repeat and ask the code

```
Enter The captcha correctly 2313:
>>3124
trail1: incorrect!, try again
Enter The captcha correctly 2313:
>>4124
trail2: incorrect!, try again
Enter The captcha correctly 2313:
>>4124
trail3: incorrect!, try again
Enter The captcha correctly 2313:
>>421
trail4: incorrect!, try again
Enter The captcha correctly 2313:
>>4124
trail5: incorrect!, try again
try again,next time.
PS C:\Users\Nathan Hailu\Desktop>
```

Break time

1) Mikiyas is a GTST company owner, and wanted to create a program that validates users so tried to make a login page can u help him?
   a) If the login is failed it have to ask again 5 times then display “ Sorry u are limited!”
   b) Accept 2 inputs from user (username and password)
   c) Validates it will dictionaries. -> change the list to dic
      i) users = ['Nathan',2313,'Geez','pass231','Abebe',092313133,'Miki',"pl3s34D0n'tH4ckM3"]
   d) Display “ Welcome to GTST Company!” if success else “Incorrect Login!”
   e) HINT: use nested things only!

Class is over

1) DO notes
2) Practice the codes
3) Do more challenges if you need challenges i will give u more on telegram.