In [32]:

```
from IPython.display import Image
Image(filename='logo.png', height=340, width=900)
```

Out[32]:


# 03 - CONTROL FLOW STATEMENTS & LOOPS

## CONTROL FLOW STATEMENTS

When you run code in Python, each statement is executed in the order in which they appear. Programming languages like Python let you change the order in which code executes, allowing you to skip statements or run certain statements over and over again. Programming constructs that let you alter the order in which code executes are known as **control flow statements**.

## IF STATEMENT

The most basic control flow statement in Python is the **`if`** statement. An if statement checks whether some logical expression evaluates to true or false and then executes a code block if the expression is true.

In Python, an if statement starts with if, followed by a logical expression and a colon. The code to execute if the logical expression is true appears on the next line, indented from the if statement above it by 4 spaces:

In [33]:

```
x=10                                                    #Assign some variables
y=5
if x>y:                                                 #If Statement
    print("x is greater than y")
```

```
x is greater than y
```

In [34]:

```
x=10
y=5
if x>y:
    print("x is greater than y")
    print("Hello")
    print("Today is Thursday")
print("Maruti")
```

```
x is greater than y
```

```
x is greater than y
Hello
Today is Thursday
Maruti
```

In [35]:

```
x=1
y=5
if x>y:
    print("x is greater than y")
    print("Hello")
    print("Today is Thursday")
print("Maruti")
```

```
Maruti
```

In this example we use two variables, x and y, which are used as part of the if statement to test whether x is greater than y. As x is 10, and y is 5, we know that 10 is greater than 5, and so we print to screen that "x is greater than y".

### Example 1: If the number is positive, print an appropriate message

In [36]:

```
num = 3
if num>0:
    print(num, "is a positive number.")
```

```
3 is a positive number.
```

## INDENTATION

Python relies on indentation, using whitespace, to define scope in the code. If statement, without indentation will throw an error.

## ELSE STATEMENT

If statements are often accompanied by else statements. **Else statements come after if statements and execute code in the event that logical expression checked by an if statement is false:**

### Example 2: Take an input from the user to check how many days are there in a leap year. Print a congratulatory message if the answer is correct

In [37]:

```
days=int(input("How many days are there in a leap year?\n"))
if days == 366:
    print("Congratulations")
else:
    print("Please re-appear for the test")
```

```
How many days are there in a leap year?
366
Congratulations
```

### Example 3: Ask the user, "if python is an interpreted language, yes or no?". If the answer is correct send a message that he has cleared the test else send a message that he has passed the test. In either case print a Thanks message too!

In [38]:

```
answer
```

Out[38]:

Out[38]:

```
'no'
```

In [39]:

```
answer=input("Is Python an Interpreted Language? Yes or No?\n").lower()
if answer=="yes":
    print("You have cleared the Test")
else:
    print("You have failed the test")

print("Thanks for appearing")
```

```
Is Python an Interpreted Language? Yes or No?
Yes
You have cleared the Test
Thanks for appearing
```

## ELIF STATEMENT

The elif keyword is pythons way of saying **"if the previous conditions were not true, then try this condition".**

In [40]:

```
x=33
y=33
if x>y:
    print("x is greater than y")
elif x==y:
    print("x and y are equal")

else:
    print("x is less than y")
```

```
x and y are equal
```

In this example x is equal to y, so the first condition is not true, but the elif condition is true, so we print to screen that "x and y are equal".

### Example 4: Ask the user "Which Python data type is an ordered sequence?". If his answer is list or tuple, send him a test clearance message else send him a message that his input is wrong. You CANNOT use OR operator to solve this.

In [41]:

```
response=input("Which python datatype is an ordered sequence?\n").lower()
print("You entered: ", response)

if response=="list":
    print("You have cleared the test.")
elif response=="tuple":
    print("You have cleared the test.")
else:
    print("Your input is wrong. Please try again!")
```

```
Which python datatype is an ordered sequence?
No
You entered:  no
Your input is wrong. Please try again!
```

## NESTED IF STATEMENTS

Some programs may have a code block under an “if” clause which subsequently has another conditional block as the first statement.

In such a case, Python allows nesting of an if-else or if-elif-else inside another conditional clause.

Python doesn't limit the level of nested conditions in a program. Given below is the syntax of a multi-level nested if-elif-else statement.

### EXAMPLE 5: # In this program, we input a number, check if the number is positive or negative or zero and display an appropriate message. This time we use nested if

In [42]:

```
num = float(input("Enter a number: "))
if num >= 0:
    if num == 0:
        print("Zero")
    else:
        print("Positive number")
else:
    print("Negative number")
```

```
Enter a number: 25
Positive number
```

### EXAMPLE 6:

In [43]:

```
response = int(input("How many days are there in a leap year? "))
#print("You entered:", response)

if response == 366 :
    #print("You have cleared the first level.")
    response = input("What month has an extra day in leap year?? ").lower()
    if response == "february" :
        print("You have cleared the test.")
    else:
        print("You have failed the test.")

else:
    print("Your input is wrong, please try again.")
```

```
How many days are there in a leap year? 366
What month has an extra day in leap year?? february
You have cleared the test.
```

### Question 1: Take values of length & breadth of a rectangle from the user and check if it is square or not?

### Question 2: A shop will give discount of 10% if the cost of purchased quantity is more than 1000. Ask user for quantity. Suppose, one unit will cost 100. Judge and print total cost for user.

### Question 3: A company decided to give bonus of 5% to employee if his/her year of service is more than 5 years. Ask user for their salary and year of service and print the net bonus amount.

### ANSWER 1:

In [44]:

```
length=int(input("Enter Length:"))

breadth=int(input("Enter Breadth:"))

if length==breadth:
    print("Yes, it is a square")
else:
    print("No, it is a rectangle")
```

```
Enter Length:25
Enter Breadth:26
No, it is a rectangle
```

## ANSWER 2:

In [45]:

```
quantity=int(input("Enter the quantity:"))
if quantity*100>1000:
    print("Cost is: ", ((quantity*100)-(0.1*quantity*100)))
    print("Discount is: ", (0.1*quantity*100))
else:
    print("Cost is:", quantity*100)
    print("Discount is: ", 0)
```

```
Enter the quantity:15
Cost is:  1350.0
Discount is:  150.0
```

## ANSWER 3:

In [46]:

```
salary=int(input("Enter the Salary: "))
yos=int(input("No. of Years of Service: "))
if yos>5:
    print("Bonus is", 0.05*salary)
else:
    print("No Bonus")
```

```
Enter the Salary: 25555000
No. of Years of Service: 5
No Bonus
```

## RANGE

The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

In [47]:

```
for x in range(6):          # for every element in range (0 to 6-1)
    print(x)                # print every element
```

```
0
1
2
3
4
5
```

The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, 3):

In [48]:

```
for x in range(2,103,10):   #starting 2, and ending 100(becuase 101 is exclusive) (Start, Stop,
Increment)
    print(x)
```

```
2
12
22
```

```
32
42
52
62
72
82
92
102
```

### likewise it can be used in case of strings:

In [49]:

```
for letter in "Welcomel india":
    if letter == "l":
        print('Current Letter:',letter)
    else:
        print("Current Letter is not l:", letter)
```

```
Current Letter is not l: W
Current Letter is not l: e
Current Letter: l
Current Letter is not l: c
Current Letter is not l: o
Current Letter is not l: m
Current Letter is not l: e
Current Letter: l
Current Letter is not l:
Current Letter is not l: i
Current Letter is not l: n
Current Letter is not l: d
Current Letter is not l: i
Current Letter is not l: a
```

## FOR LOOP

For loops are a programming construct that let you go through each item in a sequence and then perform some operation on each one. For instance, you could use a for loop to go through all the values in a list, tuple, dictionary or series and check whether each conforms to some logical expression or print the value to the console.

In [50]:

```
# Example 1:

cars=["Maruti", "Honda", "Toyota"]
for x in cars:
    print(x)
```

```
Maruti
Honda
Toyota
```

In [51]:

```
# Example 2:

for x in "banana":
    print(x)
```

```
b
a
n
a
n
a
```

In [52]:

```
# Example 3:

mysequence=tuple(range(0,101,10))
for number in mysequence:
    print(number+100)
```

```
100
110
120
130
140
150
160
170
180
190
200
```

### Question 1: Using For Loop, Write a program which will find all such numbers which are divisible by 7 but are not a multiple of 5, between 2000 and 3200 (both included). The numbers obtained should be printed in a comma-separated sequence on a single line.

In [53]:

```
a=range(2000,3201)
z=[]
for a1 in a:
    if a1%7==0 and a1%5==0:
        a1+=1
    elif a1%7==0:
        z.append(a1)
        a1+=1

print(z)
```

```
[2002, 2009, 2016, 2023, 2037, 2044, 2051, 2058, 2072, 2079, 2086, 2093, 2107, 2114, 2121, 2128, 21
42, 2149, 2156, 2163, 2177, 2184, 2191, 2198, 2212, 2219, 2226, 2233, 2247, 2254, 2261, 2268, 2282,
2289, 2296, 2303, 2317, 2324, 2331, 2338, 2352, 2359, 2366, 2373, 2387, 2394, 2401, 2408, 2422, 242
9, 2436, 2443, 2457, 2464, 2471, 2478, 2492, 2499, 2506, 2513, 2527, 2534, 2541, 2548, 2562, 2569,
2576, 2583, 2597, 2604, 2611, 2618, 2632, 2639, 2646, 2653, 2667, 2674, 2681, 2688, 2702, 2709, 271
6, 2723, 2737, 2744, 2751, 2758, 2772, 2779, 2786, 2793, 2807, 2814, 2821, 2828, 2842, 2849, 2856,
2863, 2877, 2884, 2891, 2898, 2912, 2919, 2926, 2933, 2947, 2954, 2961, 2968, 2982, 2989, 2996, 300
3, 3017, 3024, 3031, 3038, 3052, 3059, 3066, 3073, 3087, 3094, 3101, 3108, 3122, 3129, 3136, 3143,
3157, 3164, 3171, 3178, 3192, 3199]
```

### Question 2: Go through the string below and if the length of a word is even print "even!"

st = 'Print every word in this sentence that has an even number of letters'

In [54]:

```
w=[]
st = 'Print every word in this sentence that has an even number of letters'
z=st.split()
for i in z:
    if len(i)%2==0:
        w.append(i)
print(w)
```

```
['word', 'in', 'this', 'sentence', 'that', 'an', 'even', 'number', 'of']
```

### WHILE LOOP

While loops are similar to for loops in that they allow you to execute code over and over again. For loops execute their contents, at most, a number of iterations equal to the length of the sequence you are looping over. While loops, on the other hand, keep executing their contents as long as a logical expression you supply remains true:

In [55]:

```
# Example 1:

x=5
i=0
while i<x: #print as long as the iteration(i) is less than x
    print ("India")
    i=i+1  #incrementing the iteration by 1 each time the loop executes
```

```
India
India
India
India
India
```

### QUESTION 1: Using While Loop, write a program which enables the user to extract words from a string which start from a particular alphabet (which will be inputted by the User)?

z="This is Great Guru Gram City Clean Class"

In [56]:

```
z="This is Great Gurgaon Gram City Clean Class"
x=input("Words starting with which alphabet should be extracted?\n").upper()
k=z.split()
w=len(k)
t=0
while t<w:
    if k[t][0] == x:
        print(k[t])
        t+=1
    else:
        t+=1
```

```
Words starting with which alphabet should be extracted?
G
Great
Gurgaon
Gram
```

## CONTINUE & BREAK

While loops can get you into trouble because they keep executing until the logical statement provided is false. If you supply a logical statement that will never become false and don't provide a way to break out of the while loop, it will run forever. For instance, if the while loop above didn't include the statement incrementing the value of iters by 1, the logical statement would never become false and the code would run forever. Infinite while loops are a common cause of program crashes.

The CONTINUE AND BREAK STATEMENTS work inside while loops just like they do in for loops. You can use the break statement to escape a while loop even if the logical expression you supplied is true. Consider the following while loop:

In [57]:

```
while True:  #True is always TRUE!
    print("Enjoy")
    break #but we break the Loop here
```

```
Enjoy
```

In [58]:

```
x = 100
while x < 110:
    print(x)
    if x==103:
        break
    x=x+1
```

```
100
101
102
103
```

In [59]:

```
for letter in "Welcome":
    if letter == "l":
        break
    print('Current Letter:',letter)
```

```
Current Letter: W
Current Letter: e
```

With the continue statement we can stop the current iteration, and continue with the next:

In [60]:

```
x = 100
while x < 110:
    x=x+1
    if 104<=x<=107:
        continue
    print(x)
```

```
101
102
103
108
109
110
```

In the above example, the iteration is stopped for values of x: 104,105,106 and then continues for the remaining values till x<110

In [61]:

```
for letter in "Welcome":
    if letter == "l":
        continue
    print('Current Letter:',letter)
```

```
Current Letter: W
Current Letter: e
Current Letter: c
Current Letter: o
Current Letter: m
Current Letter: e
```

In [62]:

```
for letter in 'Python':
    if letter == 'h':
        pass

    print ('Current Letter :', letter)

print ("Good bye!")
```

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
Good bye!
```

## PASS

You can consider python pass statement as no operation statement. The difference between Python comments and pass statement is; comments are being eliminated while interpreting the program but pass statement does not. It consumes execution cycle like a valid statement. For example, for printing only the odd numbers from a list, our program flow will be

List <- a list of number for each number in the list: if the number is even, then, do nothing else print odd number

In [63]:

```
#Generate a list of number
numbers = [ 1, 2, 4, 3, 6, 5, 7, 10, 9 ]
#Check for each number that belongs to the list
for number in numbers:
        #check if the number is even
 if number % 2 == 0:
                #if even, then pass ( No operation )
                pass
 else:
                #print the odd numbers
  print (number)
```

```
1
3
5
7
9
```