Control Flow Structures

UNIT - II

By: N. R. TRIVEDI

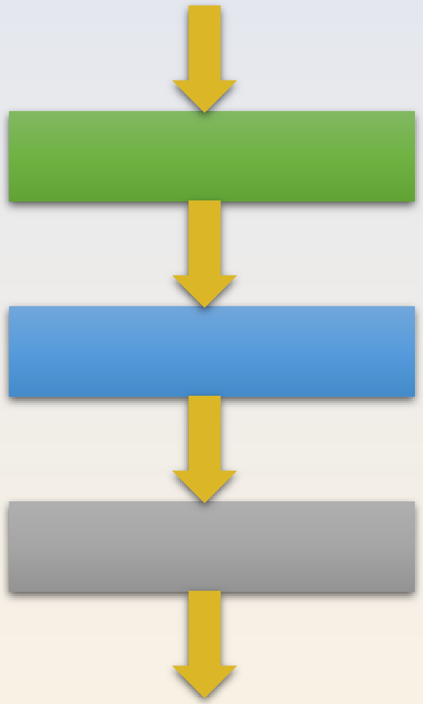
Unit-2: Control Flow Structures

CO2 - Apply control flow structures to solve the given problems.

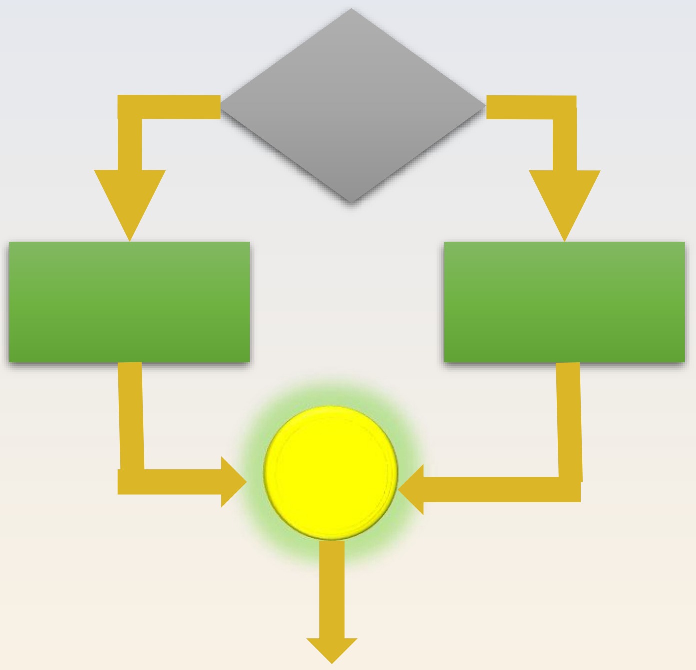
- Explain different types of Control Structures
- Develop programs using Decision making Structures
- Develop programs using Loops

2.1 Introduction to Control Sturctures 2.2 if, if-else statements 2.3 Nested if-else and if-elif-else statements 2.4 switch (match) statement	Decision Making Structures
2.5 for loop 2.6 while loop 2.7 Nested loops 2.8 break, continue and pass statements	Loops

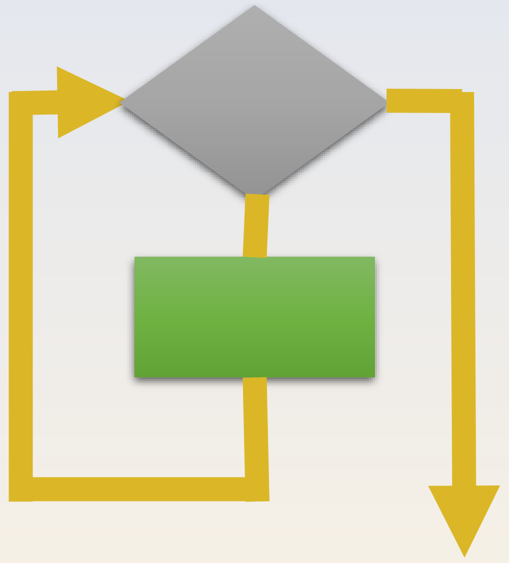
Introduction to Control Structures



Sequence



Selection



Iteration

Types of Control Structures

- It is a part of Structured Programming
- It includes features like:

- Sequence
- Selection
- Iteration or Looping
- Branching or Jumping Statements

Sequence Statement

- Default and most basic control structure of program which executes the program line by line starting from 1st line. Executing each line in sequence.

```
Statement 1
Statement 2
Statement 3
......
......
......
```

```
num1 = 1.5
num2 = 6.3
sum = num1 + num2
print('The sum is =' + sum)
```

Performs Addition with numbers

Performs Concatenation with Strings

Selection Statements

- Selection statement changes the flow of program from sequential to a specific flow based upon the result of some condition / expression.
- Based upon the result some chosen block of code is executed.
- These statements are:
  - if, if-else statements
  - Nested if-else and if-elif-else (Ladder else-if) statements
  - switch (match) statement

Selection Statements: if, if-else

- Syntax:

if Syntax

```
if (conditions):
    #Statements if True
#Next Statements outside if
```

if-else Syntax

```
if (conditions):
    #Statements if True
else:
    #Statements if False
#Next Statements outside if
```

- Example:

```
a = 30
b = 20
if (a > b) :
    print(a)

Output: 30
```

```
a = 50
b = 70
if (a > b) :
    print(a)
else:
    print(b)

Output: 70
```

Block Management

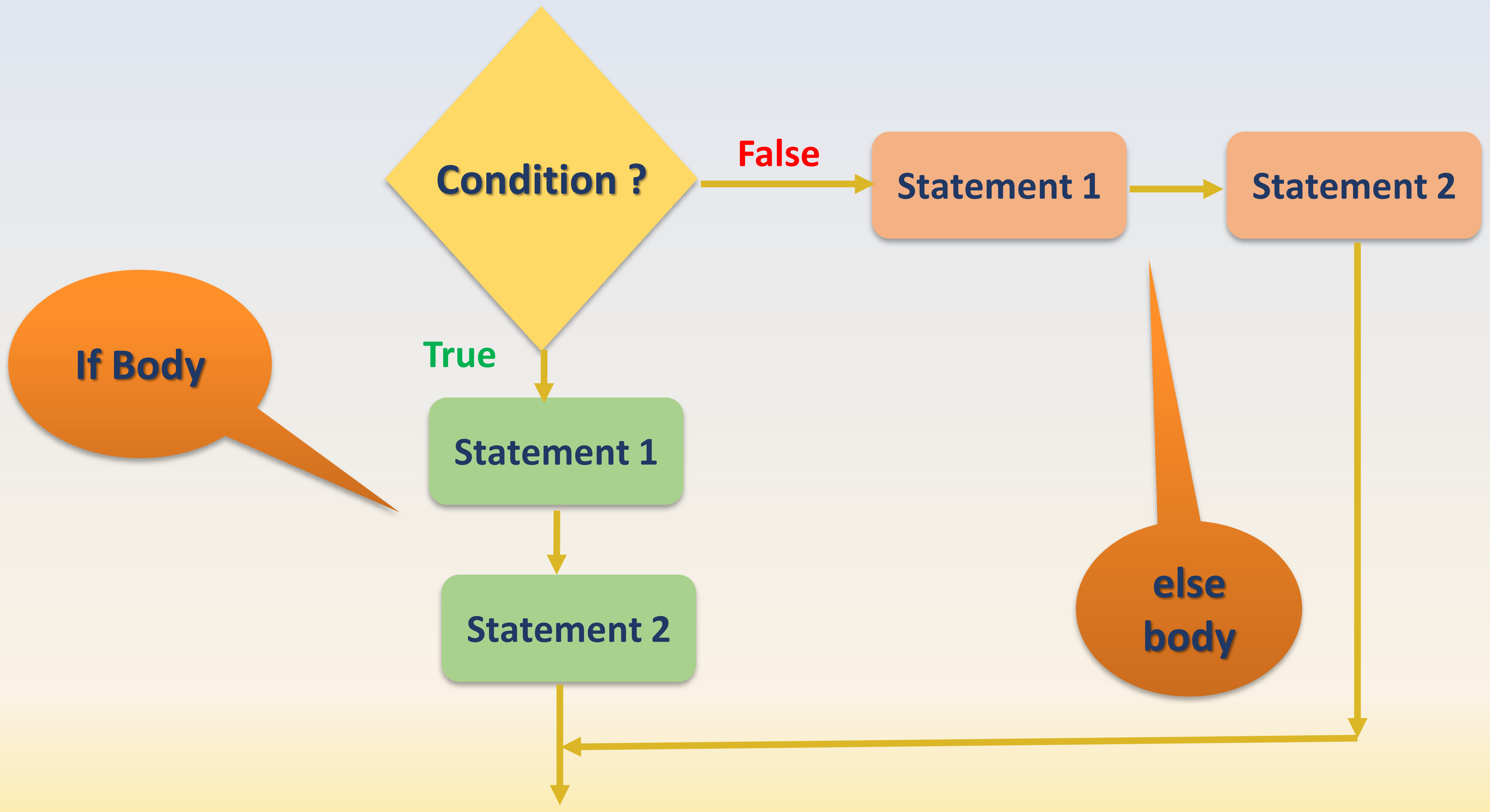
```
a = 50
b = 70
if a > b :
    print(a)
else :
    print(b)

Output: 70
```

- Python does not support parenthesis for blocks so all the statements having block structures ends with a colon (:)
- Also in the next line you need to provide a proper indent
- if parenthesis are optional if only 1 condition is present

- When you get back to the old indent Python interpreter ends the block

- *Note: Indentation is necessary in python and it must be of same no of spaces / tabs every time, otherwise Python interpreter will get confused and gives an error.*

Control Flow for if-else Statement



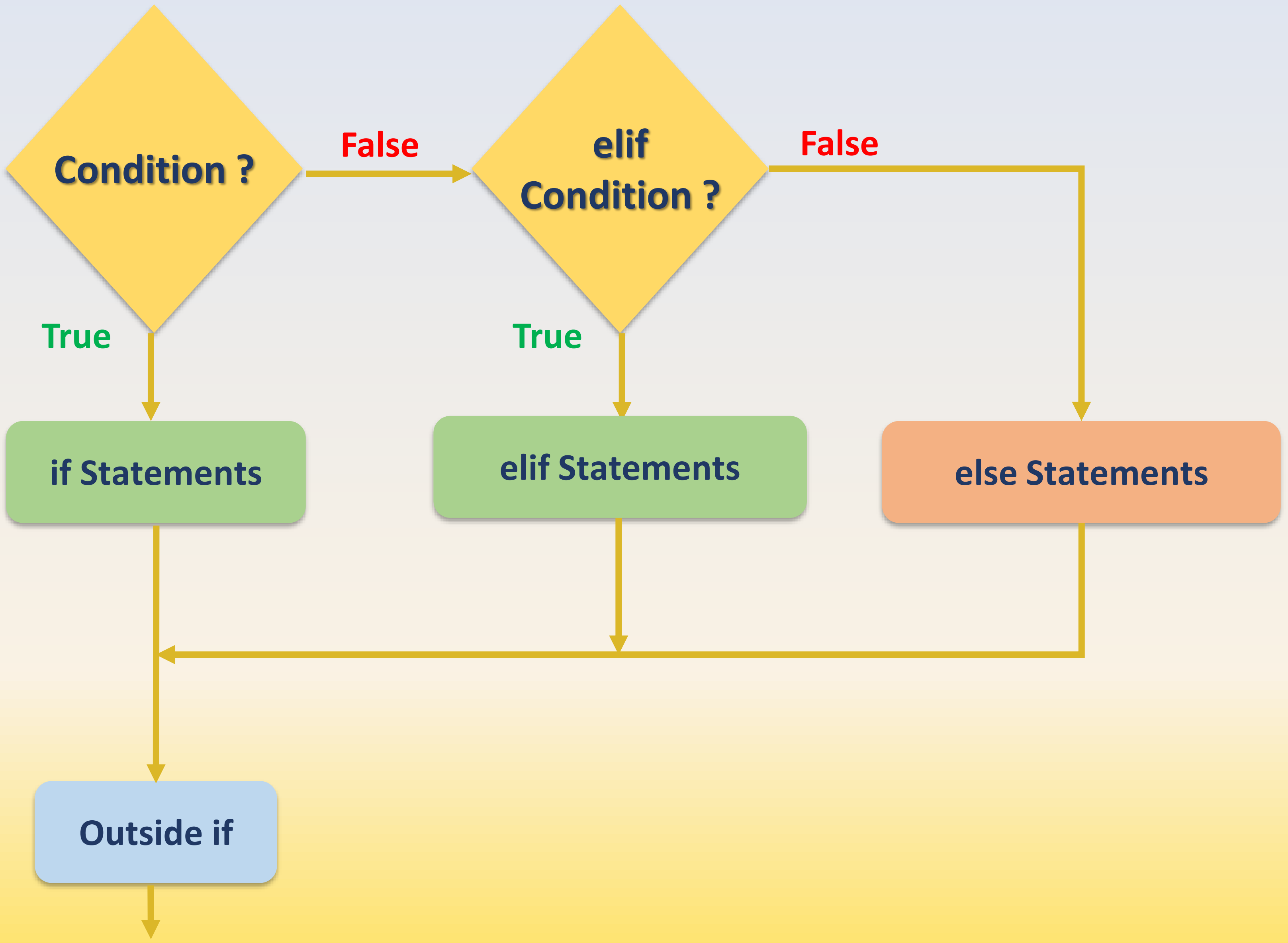
Nested if-else and if-elif-else (Ladder else-if)

- An if statement inside an if is known as Nested if statement.
- if-elif (else if)-else where else is ended with another if is known as ladder else-if.
- Any number of these statements can be nested inside one another.
- Indentation gives the nesting level and forms the block.
- Too much nesting can be confusing because there's no parenthesis to define block so we need to be extra careful while using too much nesting

```
if test_expression:
    #Body of if
elif test_expression:
    #Body of elif
else:
    #Body of else
```

$\Bigg\}$ if-elif-else Statement Syntax

Control flow for if-elif-else



elif Example

```
var = 100
if var < 200:
   print("Expression value is less than 200")
   if var == 150:
      print("Which is 150")
   elif var == 100:
      print("Which is 100")
   elif var == 50:
      print("Which is 50")
   elif var < 50:
      print("Expression value is less than 50")
else:
   print("Could not find true expression")

print("Good bye!")
```

Output:

```
   Expression value is less than 200
   Which is 100
   Good bye!
```

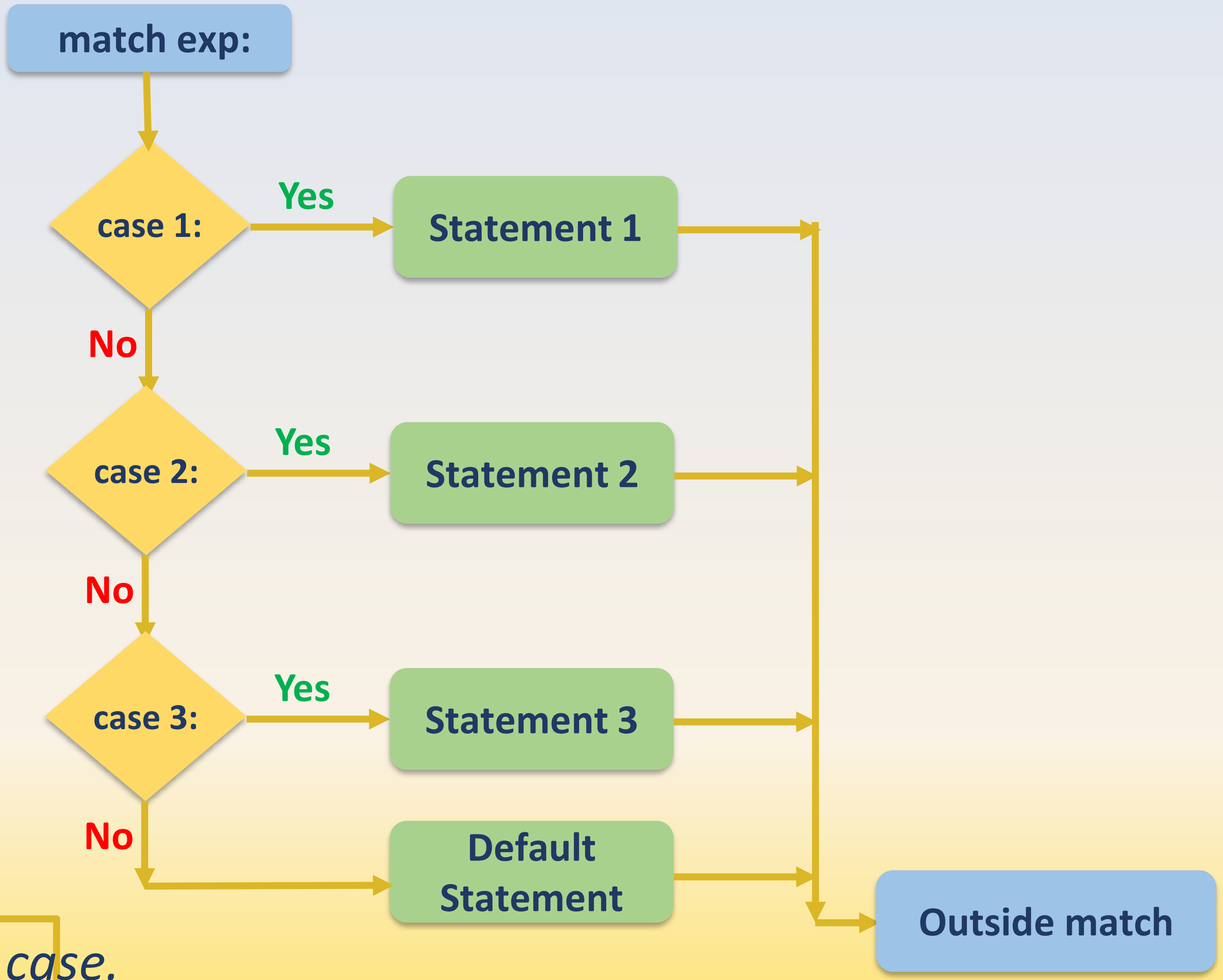
Control Structures: switch (match)

- Switch statements are introduced in Python 3.10 as match-case statement to form a “Structural Pattern Matching”.
- Prior to that version we had only elif ladder to implement switch.
- match and case are known to Python interpreter like other keywords but still you can use variables of those names. So those keywords are known as “Soft Keywords” in Python because they are still not listed in help('keywords’) function.

match-case syntax

```
match choice_var:
    case pattern-1:
        #action-1
    case pattern-2:
        #action-2
    case pattern-3:
        #action-3
    case _: / case other:
       #action-default
```

_ or other is written to specify the default case.



match-case Example

```python
lang = input("What's the programming language you want to learn? ")
match lang:
    case "JavaScript":
        print("You can become a web developer.")
    case "Python":
        print("You can become a Data Scientist")
    case "PHP":
        print("You can become a backend developer")
    case "Solidity":
        print("You can become a Blockchain developer")
    case "Java":
        print("You can become a mobile app developer")
    case _:
        print("The language doesn't matter, what matters is solving problems.")
```

Iteration / Loops

- Loop can execute a block of code number of times till some condition is met (True).
- Loops in Python:
  - while Loop
  - for Loop
  - Nested Loops

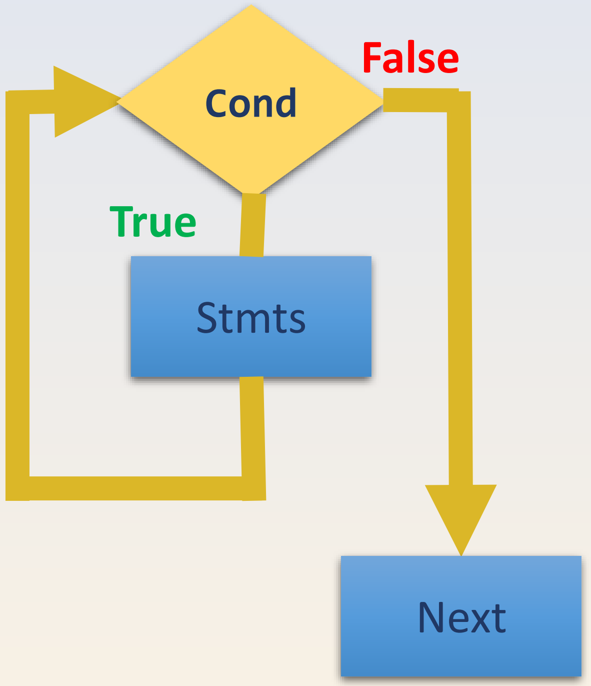
- *Note: do-while loop is not available in Python unlike C/C++.*
- *Note: Functioning of for Loop is a bit different than that of what we have already seen in C/C++.*

Loops: while

- A while loop executes a block of code repeatedly based upon testing of some boolean condition.
- Syntax:

```
while (condition(s)):
        #Loop statements
#Next Statements
```

Again, colon (:) is must to identify block statement



- Loop statements are executed until the condition(s) become False.
- After that Next Statements are executed.
- If the condition becomes false from the very 1st time then Loop won’t be executed at all.
- Example: Print Natural nos, Sum of Natural nos, Fibonacci Sequence etc.

range() function

- range() function is used in Python to get some series of natural numbers.
- Syntax: `range ([start,] stop [, step])`
- Function takes 3 arguments from which only stop is necessary, other 2 are optional. Here the series will be printed [start, stop), stop no excluded.
- Examples:

```
range (10)
```

Will give series from 0 to 9 (default start is 0).

```
range (10,20)
```

Will give series from 10 to 19.

```
range (10,20,2)
```

Will give series from 10 to 19 (increment will be now 2).
means 10, 12, 14, 16, 18

Loops: for

- for Loop in Python is similar to the for-each loop present is some languages.
- Its more convenient than while when we have to iterate through each elements of some sequence / series of elements.
- Syntax:

```
for  element  in  iteratable:
          #Loop statements

#Next Statements
```

Examples:

```
for     i     in     range(10,20):
          print(i, end=",")
print("\nLoop ended.")
```

Output:

```
10,11,12,13,14,15,16,17,18,19,
Loop ended
```

Loops: for

- IteratingaList:

```
numbers = [23,56,36,42,85,62]
for i in numbers:
     print(i, end=",")
print("\nLoop ended.")

Output:
  23,56,36,42,85,62
  Loop ended
```

Traversing a String:

```
str1 = 'abcdefgh'
for i in str1:
     print(i, end=",")
print("\nLoop ended.")

Output:
  a,b,c,d,e,f,g,h
  Loop ended
```

break, continue Statements

- Usually these statements are used in Loops.
- When break statement is executed it will terminate the current executing block of Loop and directly executes the next statement available after the loop block.
- Example:

```
for i in range(10,20):
      print(i, end=",")
      if i == 15:
            break
print("\nLoop ended.")
```

The loop will be terminated once this statement is executed at i == 15.
Examine Output

Output:

```
  10,11,12,13,14,15,
  Loop ended
```

break, continue Statements

- On the other hand continue statement does not break the whole loop but just skips the current iteration of the loop after the continue statement and loop moves on to the condition part for the next iteration.
- Example:

```
for i in range(10,20):
     if i == 15:
          continue
      print(i, end=",")
print("\nLoop ended.")
```

The iteration will be skipped once this statement is executed at i == 15.
Examine Output

Output:

```
  10,11,12,13,14,16,17,18,19,
  Loop ended
```

pass Statement

- pass statement is used when there is some statement required to be executed syntactically but you don't want any code or logic to be executed there. So you can simply write pass in place of that.
- pass Statement does nothing.
- Example:

```
i = input("Enter a String: ")
if i == 'a':
      pass
else:
      print('Else Executed')
```

- Here we don't want anything to execute if the value was 'a', otherwise we need something to be executed.
- But syntax of if requires something to be written in if block so we can write pass there.

break v/s continue

break	continue
It terminates the execution of remaining iteration of the loop.	It terminates only the current iteration of the loop.
'break' resumes the control of the program to the end of loop enclosing that 'break'.	'continue' resumes the control of the program to the next iteration of that loop enclosing 'continue'.
It causes early termination of loop.	It causes early execution of the next iteration.
'break' stops the continuation of loop.	'continue’ does not stop the continuation of loop, it only stops the current iteration.