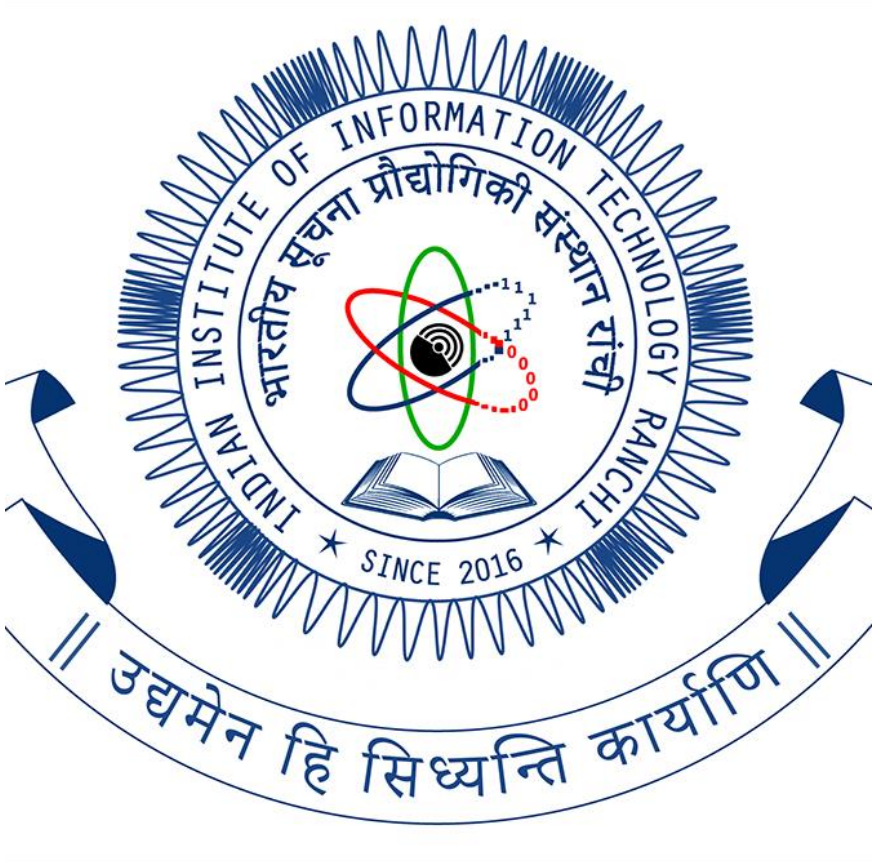
Python Programming

CS2001

Lecture-3: Flow Control statements



python™

- **Decision Control Instructions (or Conditional flow)**
  - a block of code is executed based on specific condition
- **Loop Control Instructions (Iterative flow)**
  - a block of code is repeatedly executed

If…else statement

- **Syntax**

```
if (condition or bool expression):
    #code block for True condition
    …
else:
    #code block for False condition
    …
Additional statements, if any…
```

Exercise:
Write python code to check whether the number is odd or even?

If statement without else clause

- **Syntax**

```
if (condition or bool expression):
    #code block for True condition
    …
Additional statements, if any…
```

If…elif…else statement

- **Syntax**

```
if (condition-1 or bool expression-1):
    #code block for True condition-1
    …
elif condition2:
    #code block for True condition-2
…
else:
    #code block for False condition
    …
Additional statements, if any…
```

Expression within if

The conditional expression given within the if () can be

- Comparison expressions
- Logical expressions
- Membership expressions
- Identity expressions

Comparison expressions

- **`A > B`**
  - returns True if and only if the actual value of A is greater than B
- If A and B are strings,
  - the case of the letters will determine True or False
  - Uppercase letters have lower ASCII value than the lowercase.
    - **`'Python'> 'python'`** is evaluated as False
    - **`'Python'> 'PYthon'`** is evaluated as True

Logical expressions

- To combine two or more comparison expressions
  - `A > B and A > C` evaluates to TRUE if A is greater than both B and C
  - `A > B and B > C` can be simplified as A > B > C in python
  - `A < B and A < C` can be simplified as A < B < C
  - `A > B or A > C` evaluates to TRUE if either A is greater than B or A is greater than C
  - `not X > 2` evaluates to TRUE if X is less than 2

Membership and identity expressions

- In, not in , is , is not can also be used within if clause.
- They will also return True or False

```
>>> S= "COMPUTER"
>>> T="computer"
>>> I='C'
>>> I in S ??
>>> 'A' in S ??
>>> S is T ??
```

All() and any()

- If there are a number of expressions to be combined with **and**
- we can use built in function all()
- Similarly, instead of using **or** expressions, we can apply any()
- Both all() and any() can take only one argument :- could be a list or tuple
- Example:
    - instead of writing

```
if(a>0 and b>0 and c>0 and a+b+c==100):
```

    - we can use

```
if(all((a>0,b>0,c>0,a+b+c==100))):
```

Nested if…else statement

- If there's a need to check multiple conditions
- You can nest if…elif..else within another if…elif…else statement
- **Syntax**

```
if (condition-1 or bool expression-1):
    if condition_1:
        statements
    else:
        statements
elif condition2:
    #code block for True condition-2
…
else:
    #code block for False condition
    …
Additional statements, if any…
```

Single line conditional expressions

- Syntax:
  - expression1 if condition else expression 2
  - Example:

```
Temp=int(input("enter temperature in Celsius\n"))
Status="Hot" if Temp > 35 else "Normal"
```

- Nesting:

```
Status="Hot" if temperature > 35 else "Cold" if temperature
<20 else "Normal"
```

Loop control instructions

- while
- while with else
- for
- for with else
- Nested for loops

While Loop

- Similar to that of *C*
- Slight modification in the syntax

```
while (condition):
    statements
```

- Condition is like that in decision control statements

While Loop Exercise

- display all the digits from 0 to 10 in ascending order
- 10 to 0 in descending order
- Strip the first 2 characters of a string and print result
  - E.g. `SaReGaMaPa`
    `ReGaMaPa`
    `GaMaPa`
    `MaPa`
    `Pa`

break

- 'break' is used to exit from the loop (or nested loop)
- Exits only from the loop which has break statement

```
subj = 'Python"
x = 0
while x < len(subj) - 1:
    if (subj[x] == 'n':
        break # this will end while loop prematurely
    print(subj[x])
    x +=1
```

continue

- 'continue' is used to bypass the remaining statements in the loop and to go back to the beginning of the loop

```
myStr = input('Enter a string')
x = -1
newStr = ''
while x < len(myStr) - 1:
    x = x + 1
    if (myStr[x] in ['a','e','i','o','u']):
        newStr = newStr + '#'
        continue # control jump back to while loop
    newStr = newStr + myStr[x]
print(newStr)
```

pass

- Pass is just a null statement
- It is a placeholder to tell interpreter to do nothing
- How is it different than comment??
- if(condition):

```
    #without any statements is invalid in Python
Else:
    pass
```

- Similarly, while(condition): without any code block will give you syntax error

while with else

- The additional feature of while statement in Python is the inclusion of else block

```
while(condition):
    statements
else:
    statements
```

- Else block will be executed when the condition given in while turns to be False. (Upon completion of the iterations of while loop)

while with else

```
a = int(input("Enter a number"))
while(a % 2 == 0):
   print(a, 'is even')
else:
   print(a, 'is odd')
```

??

break

Infinte loop

```
while True:
    age = input("Enter your age in years:")
    if age.isdigit():
        print('you are', age, 'years old')
        break
    print('Age is counted in numbers bro !!')
```

For loop

For loop is used to step through any sequence or a set of things, such as a string, a list or a tuple

- for loop is a **definite loop** whereas a while loop is an **indefinite loop**
- A while loop runs till a condition becomes False
- For loop runs as many times as there are items in the set
- Syntax:

```
for each_item in mySet:
   #do something
else:
   #do something else
```

For loop

Loop through each character in a string using for loop

```
myStr = 'PYTHON'
for i in myStr:
    print(i)
```

```
Output:
P
Y
T
H
O
N
```

Nested for loop

Nested for Loop good for comparing two collections, analysing

multidimensional array

```
s1 = input('First String')
S2 = input('Second string')
for i in s1:
    for j in S2:
        if i ==j:
        print(i, 'matched')
```

for loop with else

- Like while statement for loop can also include an else block
- Else block will be executed upon the condition in for loop becomes invalid.
- Usually for with else may have if() and break within for loop

```
for j in S:
      if(condition):
            #do this
            break
else:
      …
```

Hints for usage of 'while' & 'for' loops

- for loop:
  - When you're aware of max number of times that you'll need to execute the body of the loop
  - Or if you have a collection (e.g. string, list, dict, etc.) and you want to go over individual element
- while loop:
  - You need to repeatedly do some computation until some condition is met, and you have no idea when this will happen

Range function

- Not a control flow statement, but range() fxn helps very often in the construction of a loop (mostly for loop)
- range([start], stop[,step])

  Note: [] means parameter is optional
  - Default value for start = 0
  - Default for step = 1
- range(10) = range(0,10) = range(0,10,1)
- Range(n) : creates sequence from 0 to n-1 with a step of 1
- Important to note that it accepts only int arguments (+ve/-ve), no float

Enumerate

- To see the index of each element, built in function enumerate can be used in for loop

```
s = 'python'
for j in enumerate(s):
    print(j)
```

OUTPUT

```
(0, 'p')
(1, 'y')
(2, 't')
(3, 'h')
(4, 'o')
(5, 'n')
```

```
s = 'python'
for i, j in enumerate(s):
    print(i, j)
```

```
0 p
1 y
2 t
3 h
4 o
5 n
```

Exercise

- Write a python script to check if there's any vowels in the given string?
  - Take a string as an input from user
  - Create a list/string of vowels (Cap + small)
  - Apply a for loop over input string
    - And check each element if its is in the list/string of vowels
    - If True => apply a break
  - Else => No vowels.