MODULE II

Control Structures in Python:

Conditional Statements - if statements, if-else statement, nested if-else statement, Syntax and executions, Looping statements -For and For with else, While and While with Else, Syntax and executions. Control Statements - break, continue and pass – Syntax and executions.

Python Function

Design with functions: hiding redundancy, complexity ,basic syntax , scope of variables, arguments and return value, formal vs. actual arguments ,types of function , variable function arguments – default argument ,keyword argument ,arbitrary argument , recursion

RA20 APTT

Conditional Execution or Statements

- All the programs in the preceding Unit execute exactly the same statements regardless of the input (if any) provided to them.
- They follow a linear sequence: Statement 1, Statement 2, etc. until the last statement is executed and the program terminates.
- Linear programs like these are very limited in the problems they can solve.
- This Unit introduces **constructs** that allow program statements to be **optionally executed,** depending on **the context of the program's** execution.

RA20 APTT

Using spaces and tabs in python editor

- It is important not to mix spaces and tabs when indenting statements in a block.
- In many editors you cannot visually distinguish between a tab and a sequence of spaces.
- The number of spaces equivalent to the spacing of a tab differs from one editor to another.
- Most programming editors have a setting to substitute a specified number of spaces for each tab character.
- For Python development you should use this feature.
- It is best to eliminate all tabs within your Python source code
- How many spaces should you indent?
- Python requires at least one, some programmers consistently use two, four is the most popular number, but some prefer a more dramatic display and use eight.
- **A four space indentation for a block is the recommended Python style.**

RA20 APTT

Boolean Expressions

- Arithmetic expressions evaluate to numeric values; a Boolean expression, sometimes called a *predicate*, may have only one of two possible values: *false or true.*
- The simplest Boolean expressions in Python are *True and False*.

```
>>> True
True
>>> False
False
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

RA20 APTT

Boolean Expressions

Expression	Meaning
$x == y$	True if $x = y$ (mathematical equality, not assignment); otherwise, false
$x < y$	True if $x < y$; otherwise, false
$x <= y$	True if $x \leq y$; otherwise, false
$x > y$	True if $x > y$; otherwise, false
$x <= y$	True if $x \geq y$; otherwise, false
$x != y$	True if $x \neq y$; otherwise, false

Expression	Value
10 < 20	True
10 >= 20	False
x < 100	True if x is less than 100; otherwise, False
x != y	True unless x and y are equal

RA20 APTT

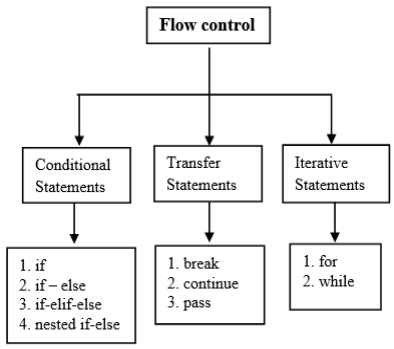
Example : Boolean Expressions

```
# Assign some Boolean variables
a = True
b = False
print('a =', a, ' b =', b)
# Reassign a
a = False;
print('a =', a, ' b =', b)
```

```
a = True b = False
a = False b =False
```

RA20 APTT

Flow control categories



RA20 APTT

Simple if statement & if/else.

- **if** statement allows code to be optionally executed

```
# Get two integers from the user
dividend, divisor = eval(input('Please enter two numbers to divide: '))
# If possible, divide them and report the result
if divisor != 0:
    print(dividend, '/', divisor, "=", dividend/divisor)
```

```
Please enter two numbers to divide: 32, 8
32 / 8 = 4.0
```

```
dividend = int (input("Enter Dividend : "))
divisor = int(input("Enter Divisor : "))
if dividend !=0:
    print(dividend, '/', divisor, "=", dividend / divisor)
```

```
PS C:\Users\home> & C:/Users/home/anaconda3/python.exe "d:/SRU/Python Programs/first.py"
Enter Dividend : 32
Enter Divisor : 8
32 / 8 = 4.0
PS C:\Users\home>
```

RA20 APTT

Cont..

Python requires the block to be indented. for example, the following if statement that optionally assigns y

```
if x < 10:
    y = x
```

could be written as

```
if x < 10: y = x
```

but may not be written as

```
if x < 10:
y = x
```

because the lack of indentation hides the fact that the assignment statement is optionally executed. Indentation is how Python determines which statements make up a block.

RA20 APTT

If with multiple statement

```
# Get two integers from the user
dividend, divisor = eval(input('Please enter two numbers to divide: '))
# If possible, divide them and report the result
if divisor != 0:
    quotient = dividend/divisor
    print(dividend, '/', divisor, "=", quotient)
print('Program finished')
```

RA20 APTT

The if/else Statement

- The if statement has an optional else clause that is executed only if the Boolean condition is false.

```
# Get two integers from the user
dividend, divisor = eval(input('Please enter two numbers to divide: '))
# If possible, divide them and report the result
if divisor != 0:
    print(dividend, '/', divisor, "=", dividend/divisor)
else:
    print('Division by zero is not allowed')
```

```
Please enter two integers to divide: 32, 0
Division by zero is not allowed
```

RA20 APTT

Nested if/else Statement

- The statements in the block of the **if or the else** may be any Python statements, **including other if/else statements.**
- These **nested if statements** can be used to develop arbitrarily complex control flow logic.

```
value = eval(input("Please enter an integer value in the range 0...10: ")
if value >= 0:          # First check
    if value <= 10:     # Second check
        print("In range")
print("Done")
```

Nested Condition (using and logic)

```
value = eval(input("Please enter an integer value in the range 0...10: ")
if value >= 0 and value <= 10: # Only one, more complicated check
    print("In range")
print("Done")
```

RA20 APTT

Cont…

```
value = eval(input("Please enter an integer in the range 0...5: "))
if value < 0:
    print("Too small")
else:
    if value == 0:
        print("zero")
    else:
        if value == 1:
            print("one")
        else:
            if value == 2:
                print("two")
            else:
                if value == 3:
                    print("three")
                else:
                    if value == 4:
                        print("four")
                    else:
                        if value == 5:
                            print("five")
                        else:
                            print("Too large")
print("Done")
```

RA20 APTT

Need of elif statement

```
value = eval(input("Please enter an integer value in the range 0...10: ")
if value >= 0:        # First check
    if value <= 10:   # Second check
        print("In range")
print("Done")
```

```
x, y, z = eval (input("Enter Three Numbers : "))
if x > y:
  if x > z:
    print("X is Greater ")
  else:
    print("Z is Greater")
elif y > z:
  print("Y is Greater ")
else:
  print("Z is Greater")
```

RA20 APTT

Cont…

```
value = eval(input("Please enter an integer in the range 0...5: "))
if value < 0:
    print("Too small")
elif value == 0:
    print("zero")
elif value == 1:
    print("one")
elif value == 2:
    print("two")
elif value == 3:
    print("three")
elif value == 4:
    print("four")
elif value == 5:
    print("five")
else:
    print("Too large")
print("Done")
```

The word `elif` is a contraction of `else` and `if`; if you read `elif` as *else if*, you can see how the code fragment

```
else:
    if value == 2:
        print("two")
```

in Listing 4.11 (`digittoword.py`) can be transformed into

```
elif value == 2:
    print("two")
```

RA20 APTT

Looping statements

- Iteration repeats the execution of a sequence of code.
- Iteration is useful for solving many programming problems.
- Iteration and conditional execution form the basis for algorithm
- construction.
- The process of executing the same section of code over and over is known as **iteration, or looping.**
- Python has two different statements, **while and for**, that enable iteration.

There are two types of iteration:

Definite iteration, in which the number of repetitions is specified explicitly in advance.

Indefinite iteration, in which the code block executes until some condition is met. In Python, indefinite iteration is performed with a **while loop**.

RA20 APTT

FOR Loop

- A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
- With the **for loop** we can execute a set of statements, once for each item in a **list, tuple, set etc.**

SYNTAX FOR LOOP

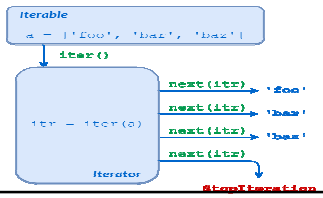
for <var> in <iterable> :
    <statement(s)>

- **<iterable>** is a collection of objects—for example, a list or tuple.
- The loop **variable <var>** takes on the value of the next element
- in <iterable> each time through the loop.

RA20 APTT

FOR Loop

Term	Meaning
Iteration	The process of looping through the objects or items in a collection
Iterable	An object (or the adjective used to describe an object) that can be iterated over
Iterator	The object that produces successive items or values from its associated iterable
`iter()`	The built-in function used to obtain an iterator from an iterable



RA20 APTT

FOR Loop

Python does the following:

•Calls **iter()** to obtain an iterator for a

•Calls **next()** repeatedly to obtain each item from the iterator in turn.

•Terminates the loop when **next()** raises the Stop Iteration exception

looping Through String

2) for x in "banana":

print(x)

RA20 APTT

Altering Loop Behaviour

Loop in Through List

fruits = ["apple", "banana", "cherry"]

1) for **x in fruits:**
    print(x)

2) for **x in fruits:**
    print(x)
    if x == "banana":
        break

3) for **x in fruits:**
    if x == "banana":
        break
    print(x)

•for **i in ['foo', 'bar', 'baz', 'qux']:**
    if 'b' in i :
        break
    print(i)

Output : foo

•for **i in ['foo', 'bar', 'baz', 'qux']:**
    if 'b' in i :
        continue
    print(i)

Output : foo , quz

RA20 APTT

Pass Statement

•for **i in ['foo', 'bar', 'baz', 'qux']:**
    pass
print(i)

In Python programming, the **pass statement** is a **null statement.**

The difference between a **comment and a pass statement** in Python is that while the interpreter **ignores a comment** entirely, **pass is not ignored.**

However, nothing happens when the pass is executed. It results **in no operation (NOP).**

RA20 APTT

Iterating Through a Dictionary

D = { 'Zoo' : 1 , 'Airport' : 2, 'Sea':3 }

For **k in D:**
    print(k)

Output : Zoo Airport Sea

2) To access Values

for **k in D:**
    print (D[k])

Output : 1 , 2 ,3

D = { 'Zoo' : 1 , 'Airport' : 2, 'Sea':3 }

D.items()

Output : Zoo Airport Sea

2) To access Values

for v in D.values():
    print(v)

Output : 1 , 2 ,3

RA20 APTT

Iterating Through a Dictionary

•method items() effectively returns a list of **key/value pairs** as tuples:

•**accessing both the keys and values**

D = { 'Zoo' : 1 , 'Airport' : 2, 'Sea':3 }

for **k, v in D.items():**

print(' k =', k, 'v =', v)

Output

k = Zoo , v = 1 k = Airport , v = 2 k = Sea , v = 3

RA20 APTT

Else Clause

• **for loop and while** can have an **else** clause as well.

•The **else block** appears after the body of the loop.

•The **statements in the else block** will be executed after all iterations in a loop are completed.

• The program exits the loop only after the else block is executed.

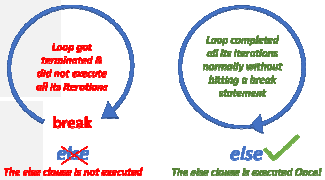
•The **else clause won't be executed** if the list is broken out of with a **break** statement

RA20 APTT

Else Clause

```
for variable_name in iterable:
    #stmts in the loop
        .
        .
        .
else:
    #stmts in else clause
        .
        .
        .

while condition:
    #stmts in the loop
        .
        .
        .
else:
    #stmts in else clause
        .
        .
        .
```



RA20 APTT

While Loop

- The **while loop in Python** is used to iterate over a block of code as long as the test expression (condition) is true.
- We generally use this loop when **we don't know the number of times** to iterate beforehand.
- The while statement checks the condition. The condition must return a Boolean value. Either True or False. **while condition: # Block of statement(s)**
- Next, If the condition evaluates to true, the while statement executes the statements present inside its block.
- The while statement continues checking the condition in each iteration and keeps executing its block until the condition becomes false.

RA20 APTT

While Loop

Python While loop

While loops **repeat the same code as long as a certain condition is true**

```
while Condition:
        statement 1
        statement 2
        …
        statement n
else:
        statement(s)
```

Expression that returns True or False

Indentation
Loop body is must be properly indented

Body of while loop
Execute as long as condition is True

Else Block (optional)
Execute only when while loop executes normally

RA20 APTT

While Loop

```
# condition: Run loop till count
is less than 5

count = 1
while count < 5:
    print(count)
    count = count + 1
```

```
# How many number div by 3
count = 0
number = 100

while number > 10:
    number = number / 3
    count = count + 1
    print('Total iteration ', count)
```

```
number = int (input('Enter any number between 100 and 300 '))
while number < 100 or number > 300:
    print('Incorrect number, Please enter correct number:')
    number = int(input('Enter a Number between 100 and 500 '))

else: print("Given Number is correct", number)
```

RA20 APTT

Range() Function

- Python **range()** function generates the immutable sequence of numbers starting from the given start integer to the stop integer.
- The **range()** is a built-in function that returns a range object that consists series of integer numbers
- range() function returns a sequence of numbers, **starting from 0** by default, and **increments by 1** (by default), and stops before a specified number.

range *(start, stop, step)*

```
x = range(6)
for i in x:
    print(i)
```

1-5

```
x = range(3, 6)
for i in x:
    print(i)
```

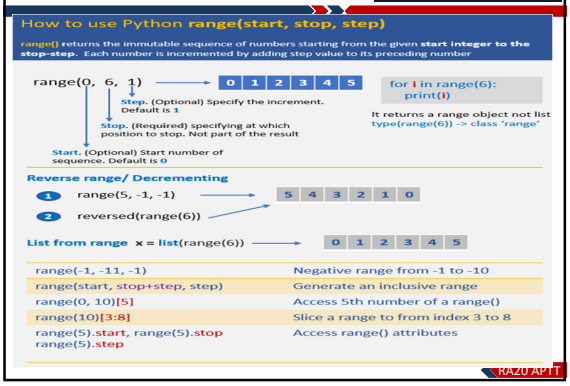
3-5

```
x = range(3, 20, 2)
for i in x:
    print(i)
```

3 5 7 9 11 ……

RA20 APTT

Range() Function



RA20 APTT

Points to remember about Range() Function

SR UNIVERSITY

- The range() function only works with the integers, So all arguments must be integers.
- No use float numbers or other data type as a start, stop, and step value.
- All three arguments can be positive or negative.
- The step value must not be zero.
- If a step=0, Python will raise a **ValueError** exception

Q. 1 Use range() to generate numbers starting from 9 to 100 divisible by 3.

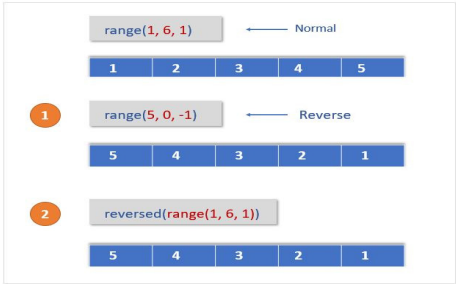
Q.2

```
1
2 2
3 3 3
```

RA20 APTT

Points to remember about Range() Function

SR UNIVERSITY



RA20 APTT

Points to remember about Range() Function

SR UNIVERSITY

```
list1 = ['Jessa', 'Emma', 20, 30, 75.5]
for i in range (len (list1) ):
    print ( list1[i] )

list1 = [10, 20, 30, 40, 50]
for i in range( len(list1) - 1, -1, -1):
    print(    list1[i],    end=" ")
```

```
for i in range(5, -1, -1):
    print(i)

for i in reversed(range(10, 21, 2)):
    print(i, end=' ')

sample_list = list(range(2, 10, 2))
print(sample_list)
```

RA20 APTT