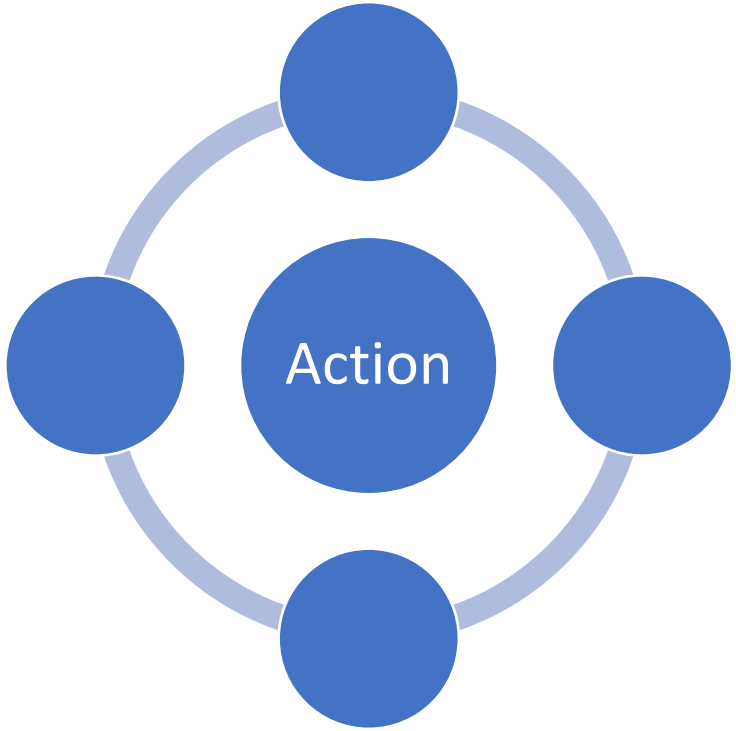
TECHNO INDIA SILIGURI INSTITUTE OF TECHNOLOGY
A Satyam Roychowdhury Initiative



While and For Loop

Repeating Actions

For and while loop

- Statements that repeat an action over and over
- While statements provides an way to code general loops
- For statement, is designed for stepping through the items in a sequence object and running a block of code for each.

While loop

- Most general iteration in Python programming language
- It repeatedly executes a block of statements as long as a test at the top keeps evaluating to a true value.
- It is called a loop because control keeps looping back to the start of the statement until the test become false.
- When the test become false , control passes to the statement that follows the while block.

General format of while loop

$$\boldsymbol{while} < \boldsymbol{test}: >:$$

$$< \boldsymbol{statesments1} >$$

$$\boldsymbol{else}:$$

$$< \boldsymbol{statements2} >$$

while statement consists of a header line with a test expression, a body of one or more indented statements, and an optional *else* part that is executed if control exits the loop without *break* statement

break statement

- Jump out of the closet enclosed loop
- Causes immediate exit from a loop

```
while True:
    name=input("enter your name")
    if name=='stop': break
    age=input("Enter your age")
    print("I am",name," and I am ",age," yr old")
```

continue statement

- It causes immediate jump to the top of a loop
- It allows you to avoid nesting

```
x=0
while x<10:
    x=x+1
    if x%2 ==0: continue
    print("x=",x)
```

Loop *else*

- It is optional
- It is executed when the loop exits without a break statement

```
num=int(input("Enter your number"))
i=2
while i<=num/2:
    if num%i==0:
        print("This is non prime number")
        break
    i=i+1
else:
    print("This is a Prime number")
```

pass statement

- It is a no-operation placeholder that is used when the syntax requires a statement, but nothing useful to say.
- A *pass* is coded to mean “to be filled in later”

```
def fun1():
    pass
```

for loop

- Generic sequence iterator in Python
- It can step through the items in any ordered sequence object.
- The for statement works on strings, list, tuple, other built-in iterables

```
for <item> in <sequence_object>:
    <statements>
else:
    <statements>
```

for statement

TECHNO INDIA SILIGURI INSTITUTE OF TECHNOLOGY

```
for i in ['subhankar','sudip','bishal']:
    print(i,end=' ')
```

```
for item in (1,'abd',10.2,'a'):
    print(item,end='  ')
```

```
d={'a':10,'b':30,'c':43}
for item in d:
    print(d[item],end=' ')
```

```
for item in "MCA":
    print(item,end=' ')
```

$range$ function

- The built-in $range$ function produces a series of successively higher integers, which can be used as indexes in a for
- $range$ is an iterator that generates items on demand

```
range([start],stop,[step])
deafault vaue of start is 0
default value of step is 1
```

```
for i in range(5,10):
    print(i,end=' ')
```

```
for i in range(10):
    print(i,end=' ')
```

```
for i in range(1,10,2):
    print(i,end=' ')
```

```
for i in range(10,0,-1):
    print(i, end=' ')
```

zip statement

- The built-in *zip* function allows to use for loops to visit multiple multiple sentences in parallel.
- *zip* takes one or more sequences as arguments and return a series of tuples that pair up parallel items taken from those sequences.

```
a=['a','b','d','t']
b=[1,2,3,4]
for (x,y) in zip(a,b):
    print(x,end=' ')
    print(y)
```

enumerate statement

- Generate both offsets and items

```
st="MCA"
for s in enumerate(st):
    print(s)

for (i,v) in enumerate(st):
    print('index=',i,end=' ')
    print('Value=',v)
```

QUIZ

- What are the main functional differences between a while and a for?
- What's the difference between break and continue?
- When is a loop's else clause executed?
- How can you code a counter-based loop in Python?
- What can a range be used for in a for loop?