Practical 2

Decision Making, Loops, Strings

Name: Chahat Kalsi
Roll no.: UE193032
Class: CSE, Section 1, 5th Sem

Decision Making

- Non-null and non-zero: True
- Null and zero: False

In [1]:

```python
val = 0
if val:
    print("Yes")
else:
    print("No")
```

```
No
```

In [2]:

```python
if None:
    print("None printed")
else:
    print("None not printed")
```

```
None not printed
```

In [3]:

```python
val = -1
lst1 = [x for x in range(-5, -1)]
lst2 = [x for x in range(-1, 5)]

if val in lst1:
    print("val in list 1")
elif val in lst2:
    print("val in list 2")
else:
    print("val not in anything")
```

```
val in list 2
```

Single Statement Suites

single if statements can go in a single lines

In [4]:

```
var = 100
if ( var  == 100 ) : print ("Value of expression is 100")
print ("Good bye!")
```

```
Value of expression is 100
Good bye!
```

Nested if statements

In [5]:

```
if True:
    if False: print("hello")
    else: print("bye")
```

```
bye
```

Loops

- while loop
- for loop
- nested loops

While Loop

Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.

In [6]:

```
lst = [x for x in range(5)]
i=0

while i<len(lst):
    print(f"List element {i} is {lst[i]}")
    i+=1
```

```
List element 0 is 0
List element 1 is 1
List element 2 is 2
List element 3 is 3
List element 4 is 4
```

In [7]:

```python
count = 0
while (count < 9):
    print('The count is:', count)
    count = count + 1
print ("Good bye!")
```

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

For Loop

Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

In [8]:

```python
for i in lst:
    print(i)
```

```
0
1
2
3
4
```

In [9]:

```python
for letter in 'Python':     # traversal of a string sequence
    print ('Current Letter :', letter)
print()
fruits = ['banana', 'apple',  'mango']

for fruit in fruits:        # traversal of List sequence
    print ('Current fruit :', fruit)

print ("Good bye!")
```

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n

Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
```

Nested Loops

In [10]:

```
for i in range(1,11):
    for j in range(1,11):
        k = i*j
        print (k, end=' ')
    print()
```

```
1 2 3 4 5 6 7 8 9 10 
2 4 6 8 10 12 14 16 18 20 
3 6 9 12 15 18 21 24 27 30 
4 8 12 16 20 24 28 32 36 40 
5 10 15 20 25 30 35 40 45 50 
6 12 18 24 30 36 42 48 54 60 
7 14 21 28 35 42 49 56 63 70 
8 16 24 32 40 48 56 64 72 80 
9 18 27 36 45 54 63 72 81 90 
10 20 30 40 50 60 70 80 90 100 
```

Loop Control Statements

- break statement
- continue statement
- pass statement

break

Terminates the loop statement and transfers execution to the statement immediately following the loop.

In [11]:

```
name = "Chahat Kalsi"
print("letters in my first name are:")
for char in name:
    if char==' ': break
    print(char)
```

```
letters in my first name are:
C
h
a
h
a
t
```

continue

Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

In [12]:

```python
print("consonants in my name are:")

for char in name:
    if char=='a' or char=='i': continue
    print(char)
```

```
consonants in my name are:
C
h
h
t

K
l
s
```

pass

The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

In [13]:

```python
for letter in 'Python':
    if letter == 'h':
        pass
        print ('This is pass block')
    print ('Current Letter :', letter)
print ("Good bye!")
```

```
Current Letter : P
Current Letter : y
Current Letter : t
This is pass block
Current Letter : h
Current Letter : o
Current Letter : n
Good bye!
```

Iterator

Iterator objects allow traversal through elements of a collection. An iterator object implements methods:

- iter()
- next()

In [14]:

```python
lst = [1,2,3,4]
it = iter(lst)
```

In [15]:

```
print(next(it))
print(next(it))
print(next(it))
```

```
1
2
3
```

In [16]:

```
lst = list('hello')
itr = iter(lst)

while True:
    try:
        print(next(itr))
    except StopIteration:
        print('finished')
        break
```

```
h
e
l
l
o
finished
```

Generator

Generator functions return iterator objects

In [17]:

```
def generatorFunct():
    yield 'a'
    yield 1
    yield 'hello'
    yield 4.32

for x in generatorFunct():
    print(x)
```

```
a
1
hello
4.32
```

In [18]:

```python
def generateNaturalNumbers(num):
    a=1
    while a<=num:
        yield a
        a+=1

x = generateNaturalNumbers(5)
print(x.__next__())
print(x.__next__())
print(x.__next__())
print(x.__next__())
print(x.__next__())
try:
    print(x.__next__())
    # throws error because only upto 5 numbers generated
except StopIteration:
    print('finished')


# iterating using for loop
print()
for i in generateNaturalNumbers(5):
    print(i)
```

```
1
2
3
4
5
finished

1
2
3
4
5
```

Strings

Declaring Strings

In [19]:

```python
# single quotes
str1 = 'hello'

# double quotes
str2 = "hello"

# tripple quotes
str3 = '''Hello
World!
Hello
Python!
'''

str4 = """
The fear
of fear
is fear
itself.
"""

print(str1)
print(str2)
print()
print(str3)
print(str4)
```

```
hello
hello

Hello
World!
Hello
Python!


The fear
of fear
is fear
itself.
```

Indexing

Python supports 0-based indexing of strings.
Negative indexing starting from the last character (-1 index) is also supported.

In [20]:

```python
str1 = "hello, world!"
print(str1[0])
print(str1[-3])
```

```
h
l
```

Slicing

In [21]:

```
# syntax: str1[start:end:skip]

print(str1[0:5])
print(str1[0:10])
print(str1[-3:-1])
print(str1[1::2])
```

```
hello
hello, wor
ld
el,wrd
```

Updating Strings

In [22]:

```
str1 = "Hello, World!"
print(str1)
str1 = str1[:5]
print(str1)
```

```
Hello, World!
Hello
```

Escape Characters

Escape characters can be used with their special backslash notations

In [23]:

```
# backspace: \b
print('CK\b')
print('---')

# tab: \t
print('Chahat\tKalsi')
print('---')

# newline: \n
print('Chahat\nKalsi')
print('---')
```

```
C
---
Chahat  Kalsi
---
Chahat
Kalsi
---
```

Raw Strings

String is printed as is. Backslash notations are treated as normal string characters.

In [24]:

```python
print('Chahat\nKalsi') # normal string
print(r'Chahat\nKalsi') # raw string
```

```
Chahat
Kalsi
Chahat\nKalsi
```

String Formatting Operator

%

In [25]:

```python
name = "Chahat Kalsi"
age = 21

print("I'm %s and I'm %d years old." % (name, age))
print("I scored %.1f in the quiz" % 9.5)
```

```
I'm Chahat Kalsi and I'm 21 years old.
I scored 9.5 in the quiz
```

String methods

In [26]:

```python
str1 = "helLo"

print(str1.capitalize())
print(str1.center(10, '-'))
print(str1.upper())
print(str1.split('l'))
print(str1.swapcase())
```

```
Hello
--helLo---
HELLO
['he', 'Lo']
HELlO
```