In [26]:

```
# transfer statements
#1. break - we can use break statments inside loops to break loop execution based on some condition.
#eg
empsalaries=[20000,40000,50000,60000]
for salary in empsalaries:
    if salary>50000:
        print("pls provide pancard to get salary")
        break
    print(salary)
```

```
20000
40000
50000
pls provide pancard to get salary
```

In [27]:

```
# 2. continue - we can use contiune statments to skip current iteration and continue next iteration.
empsalaries=[20000,40000,50000,60000,30000,25000]
for salary in empsalaries:
    if salary>50000:
        print("pls provide pancard to get salary")
        continue
    print(salary)
```

```
20000
40000
50000
pls provide pancard to get salary
30000
25000
```

In [29]:

```
# loops with else block
# Inside loop execution, if break statement not executed, then only else part will be executed.
# else means without break
empsalaries=[20000,40000,5000,6000,30000,25000]
for salary in empsalaries:
    if salary>50000:
        print("pls provide pancard to get salary")
        break
    print(salary)
else:
    print('There is no salary greater than 50000')
```

```
20000
40000
5000
6000
30000
25000
There is no salary greater than 50000
```

In [32]:

```
empsalaries=[20000,40000,5000,60000,30000,25000]
for salary in empsalaries:
    if salary>50000:
        print("pls provide pancard to get salary")
        break
    print(salary)
else:
    print('There is no salary greater than 50000')
```

```
20000
40000
5000
pls provide pancard to get salary
```

In [33]:

```
# difference between for loop and while loop ?
# we can loops to repeat code execution
# Repeat code for every item in sequence ==> forloop
# Repeat code as long as condition is true ==> whileloop
```

In [34]:

```
# how to exit from loop ==> by using break statement
# how to skip current iteration inside loop ==> by using continue statement.
# when else part will be executed wrt loop ==> if loop executed with out break
```

In [35]:

```
# pass statement - pass is a keywrd in python.
# In our programming syntatically if a block is required which won't do anything then we can define that empty bl
ock with
# pass keyword
```

In [37]:

```
#eg:
empsalaries=[10000,30000,40000,50000]
for x in empsalaries:
    if x<50000:
        print(x)
    else:
        pass
```

```
10000
30000
40000
```

In [38]:

```
# del statement- del is a keyword in python
# After using a variable it is highly recommended to delete that variable if it is no longer required, so that co
rresponding
# object is eligible for garbage collection.
empsalaries=[10000,30000,40000,50000]
for x in empsalaries:
    if x<50000:
        print(x)
    else:
        pass
del empsalaries
```

```
10000
30000
40000
```

In [40]:

```
print(empsalaries)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-40-65c83d5e238c> in <module>
----> 1 print(empsalaries)

NameError: name 'empsalaries' is not defined
```

In [41]:

```
# difference between del and none
# In the case del, the variable will be removed and we cannot access that variable.
s='aravind'
del s
print(s)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-41-76e3b3d327ab> in <module>
      3 s='aravind'
      4 del s
----> 5 print(s)

NameError: name 's' is not defined
```

In [42]:

```
# but in the case of None assignment the variable won't be removed but the corresponding object is eligible for g
arabage
# collection . Hence after assigning with None value, We can access that variable.
emp_name='aravind'
emp_name=None
print(emp_name)
```

```
None
```