Slicer

it is used to slice object

syntax [start:end:step]

In [1]:
```
a="today is december 30th and tomorrow year will end"
#end
```

In [2]:
```
a="0123456789"
a[0]
```

Out[2]: '0'

In [3]:
```
a[1:6]  #[start:end:step]
```

Out[3]: '12345'

In [4]:
```
a[::]  #[start:end:step]
```

Out[4]: '0123456789'

In [5]:
```
a[0:5:3]
```

Out[5]: '03'

In [6]:
```
a[::-1]
```

Out[6]: '9876543210'

In [7]:
```
a[::-2]
```

Out[7]: '97531'

In [8]:
```
string="today is december 30th and tomorrow year will end"
string[2:18:4]  #[start:end:step]
#2,2+4,2+4+4,2+4+4+4
```

Out[8]: 'dieb'

In [9]:
```
a="today is december aaaaaaaaa sd dasadaasd     asdfafsdasf fsdfaf asdfasfd asfdasfsf da ffasf va fva qvfa sva vfa svf asqvf aq v
#type tomorrow
a[-22:-14:1]
```

Out[9]: 'tomorrow'

Python Conditions

a == b equal

a!= b Not equal

a<b less than

a<=b less than equal to

a>b greater than

a >= b greater than equa to

a=b it means the value of b is assigned to a

Looping,

if- else statement

In [10]:
```
a=20
b=200
if b> a:
    print("b is greater than a")
print("the loop is ended, this is next line of code")
```

```
b is greater than a
the loop is ended, this is next line of code
```

In [11]:
```
a=20000
b=200
if b> a:
    print("b is greater than a")
    print("this is also under loop")
```

In [12]:
```
a=20
b=20
if b>a:
    print("b is greater than a")
elif a==b:   #else if
    print("b is equal to a")
```

```
b is equal to a
```

In [13]:
```
b=10
a=b
a
```

Out[13]:
```
10
```

In [14]:
```
a=12.0
b=12
if b>a:
    print("b is greater than a")
elif a==b:                          #else if
    print("b is equal to a")
else:
    print("a is greater than everyone")
```

```
b is equal to a
```

In [15]:
```
a=120
b=12
c=500

if a>b and c>a:
    print("both are valid")
```

```
both are valid
```

In [16]:
```
if b<a:
    pass
```

In [17]:
```
# take integer input a and b and then if a<b then cube of a else square of b
```

WHILE LOOP

it can execute a set of statements as long as any condition is true.

In [18]:
```
a=2
while a<=20:
    print(a)
    a = a+5
    # a += 5
```

```
2
7
12
17
```

Break steatment can stop the loop even if the while condition is true

In [19]:
```
a=1
while a<100:
    print (a)
    if a==5:
        break
    a=a+1
```

```
1
2
3
4
5
```

Continue

with this we can stop the current iteration and continue with next one

In [20]:
```
a=5
while a<9:
    a=a+1
    if a==7:
        continue
    print(a)
```

```
6
8
9
```

In [21]:
```
a=5
while a<9:
    a=a+1     # this space is called indentation
    print(a)
```

```
6
7
8
9
```

For loop

it is used for iterating over a sequence.

In [2]:
```
Menue="PBPMI"
for waiter in Menue:
    print(waiter)
```

```
P
B
P
M
I
```

In [3]:
```
for waiter in "PBPMI":
    print(waiter)
```

```
P
B
P
M
I
```

In [4]:
```
a="adv happy new yr guys"
for i in a:
    print(i)
```

```
a
d
v

h
a
p
p
y

n
e
w

y
r

g
u
y
s
```

In [6]:
```
a="same to you sir"
for i in a:
    b=i
    print(b)
```

```
s
a
m
e

t
o

y
o
u

s
i
r
```

In [8]:
```
list1=["name","is","not","and"]
for i in list1:
    print(i)
```

```
name
is
not
and
```

Range

range() it returns a sequence of no. starting from 0 by default and increase by 1 by default and till the value specefied .

In [10]:
```
for x in range(5):
      print(x)
```

```
0
1
2
3
4
```

In [11]:
```
range(5)
```

Out[11]:
```
range(0, 5)
```

In [12]:
```
help(range)
```

```
Help on class range in module builtins:

class range(object)
 |  range(stop) -> range object
 |  range(start, stop[, step]) -> range object
 |
 |  Return an object that produces a sequence of integers from start (inclusive)
 |  to stop (exclusive) by step.  range(i, j) produces i, i+1, i+2, ..., j-1.
 |  start defaults to 0, and stop is omitted!  range(4) produces 0, 1, 2, 3.
 |  These are exactly the valid indices for a list of 4 elements.
 |  When step is given, it specifies the increment (or decrement).
 |
 |  Methods defined here:
 |
 |  __bool__(self, /)
 |      True if self else False
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(self, key, /)
 |      Return self[key].
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __reduce__(...)
 |      Helper for pickle.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __reversed__(...)
 |      Return a reverse iterator.
 |
 |  count(...)
 |      rangeobject.count(value) -> integer -- return number of occurrences of value
 |
 |  index(...)
 |      rangeobject.index(value) -> integer -- return index of value.
 |      Raise ValueError if the value is not present.
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  start
 |
 |  step
 |
 |  stop
```

In [13]:
```
?range
```

In [14]:
```
for x in range(5,10):
      print(x)
```

```
5
6
7
8
9
```

In [15]:
```
for x in range(4,20,3):
      print(x)
```

```
4
7
10
13
16
19
```

In [19]:
```
for x in range(40,20,-3):
      print(x)
```

```
40
37
34
31
28
25
22
```

In [21]:
```
for x in range(10):
    if x==6: break
    print(x)
else:
    print("done")
```

```
0
1
2
3
4
5
```

In [32]:
```
for x in range(5):
    if x==6: break
    print(x)
else:
    print("done")
```

```
0
1
2
3
4
done
```

In [39]:
```
#NESTED for loop
a=["1","2","3"]
b=["A","B","C"]
for x in a: #1,2,3
    for y in b:  #A,B,C
        print(x,y)
```

```
1 A
1 B
1 C
2 A
2 B
2 C
3 A
3 B
3 C
```

In [48]:
```
#H.W
a="happy new year guys, this will make you good in loop"
#print without vowels aeiou

a="happy new year guys, this will make you good in loop"
b = ('a','e','i','o','u')
for x in a:
  if x in b:
    continue #Pass also works
  else:
    print(x)
```

```
h
p
p
y

n
w

y
r

g
y
s
,

t
h
s

w
l
l

m
k

y

g
d

n

l
p
```

In [50]:
```
#Write a program which will ask user to input a name, and then tell whether the name is palindrome or not
a=input("enter name")
if(a==a[::-1]):
        print("palindrome")
else:
    print("not palindrome")
```

```
enter namesns
palindrome
```

In [53]:
```
#take any integer input from the user and tell if that is an armstrong number or not
#eg 371 is an Armstrong number since 3**3 + 7**3 + 1**3 = 371
```

In [63]:
```
a= int(input("Enter a number"))

sume = 0
temp = a
t = 0

while temp>0:
  t = t+1
  temp = temp//10
print(t)
temp = a

while temp>0:
  rem = temp % 10
  sume = sume + (rem ** t)
  temp = temp //10

if a == sume:
  print("Is Armstrong")
else:
  print("No")
```

```
Enter a number4210818
7
Is Armstrong
```

In [65]:
```
"""
take integer input from user as the no. of lines.
and make a tree of stars for that many number of lines.
for eg, if input is 2 the output shall be
   *
  ***
if input is 3 then o/p is
    *
   ***
  *****
"""
```

Out[65]:
```
'\ntake integer input from user as the no. of lines.\nand make a tree of stars for that many number of lines.\nfor eg, if input is 2 the output shall be\n   *\n  ***\nif input is 3 then o/p is\n    *\n   ***\n  *****\n'
```

In [ ]:
```

```