while loop

In [1]:
```
n=int(input('enter an input'))
i=0
while i<=n:
    print(i)
    i=i+1 #i +=1
```

```
enter an input10
0
1
2
3
4
5
6
7
8
9
10
```

In [2]:
```
# reverse of a number 123 - 321
123/10
```

Out[2]:
```
12.3
```

In [2]:
```
# reverse of a number 123 - 321
n=int(input('enter value'))
rev=0
while n>0:
    r= n%10
    rev=rev*10+r
    n=n//10
print(rev)
```

```
enter value12345
54321
```

In [8]:
```
# palindrom
n=int(input('enter value'))
m=n
rev=0
while n>0:
    rev=rev*10+n%10
    n=n//10
if m==rev:
      print(m,'is palindrome')
else:
      print('not palindrome')
```

```
enter value121
121 is palindrome
```

functions

- function is a group of statments to do a specific task
- functions breaks code into small module to look more organaised

advantages of functions

- useability
- types of functions
- built in functions
- user defind function

list of builtins

In [10]:
```
dir(__builtins__)
```

Out[10]:
```
['ArithmeticError',
 'AssertionError',
 'AttributeError',
 'BaseException',
 'BlockingIOError',
 'BrokenPipeError',
 'BufferError',
 'BytesWarning',
 'ChildProcessError',
 'ConnectionAbortedError',
 'ConnectionError',
 'ConnectionRefusedError',
 'ConnectionResetError',
 'DeprecationWarning',
 'EOFError',
 'Ellipsis',
 'EnvironmentError',
 'Exception',
 'False',
 'FileExistsError',
 'FileNotFoundError',
 'FloatingPointError',
 'FutureWarning',
 'GeneratorExit',
 'IOError',
 'ImportError',
 'ImportWarning',
 'IndentationError',
 'IndexError',
 'InterruptedError',
 'IsADirectoryError',
 'KeyError',
 'KeyboardInterrupt',
 'LookupError',
 'MemoryError',
 'ModuleNotFoundError',
 'NameError',
 'None',
 'NotADirectoryError',
 'NotImplemented',
 'NotImplementedError',
 'OSError',
 'OverflowError',
 'PendingDeprecationWarning',
 'PermissionError',
 'ProcessLookupError',
 'RecursionError',
 'ReferenceError',
 'ResourceWarning',
 'RuntimeError',
 'RuntimeWarning',
 'StopAsyncIteration',
 'StopIteration',
 'SyntaxError',
 'SyntaxWarning',
 'SystemError',
 'SystemExit',
```

```
 'TabError',
 'TimeoutError',
 'True',
 'TypeError',
 'UnboundLocalError',
 'UnicodeDecodeError',
 'UnicodeEncodeError',
 'UnicodeError',
 'UnicodeTranslateError',
 'UnicodeWarning',
 'UserWarning',
 'ValueError',
 'Warning',
 'WindowsError',
 'ZeroDivisionError',
 '__IPYTHON__',
 '__build_class__',
 '__debug__',
 '__doc__',
 '__import__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'abs',
 'all',
 'any',
 'ascii',
 'bin',
 'bool',
 'breakpoint',
 'bytearray',
 'bytes',
 'callable',
 'chr',
 'classmethod',
 'compile',
 'complex',
 'copyright',
 'credits',
 'delattr',
 'dict',
 'dir',
 'display',
 'divmod',
 'enumerate',
 'eval',
 'exec',
 'filter',
 'float',
 'format',
 'frozenset',
 'get_ipython',
 'getattr',
 'globals',
 'hasattr',
 'hash',
```

```
 'help',
 'hex',
 'id',
 'input',
 'int',
 'isinstance',
 'issubclass',
 'iter',
 'len',
 'license',
 'list',
 'locals',
 'map',
 'max',
 'memoryview',
 'min',
 'next',
 'object',
 'oct',
 'open',
 'ord',
 'pow',
 'print',
 'property',
 'range',
 'repr',
 'reversed',
 'round',
 'set',
 'setattr',
 'slice',
 'sorted',
 'staticmethod',
 'str',
 'sum',
 'super',
 'tuple',
 'type',
 'vars',
 'zip']
```

user defind functions

syntax in c

```
function fname(){
    cond/stmts to execute
}
### syntax in pytthon
def frame():
    comd/stmts
    return
fname()
```

- advantages
- making of large codes into small codes
- reuse of code in function by calling its

In [13]:
```
a=1
b=10
sum([a,b])
```

Out[13]: 11

In [14]:
```
a=[1,2,3,4,5]
max(a)
```

Out[14]: 5

In [16]:
```
min(a)
```

Out[16]: 1

In [17]:
```
len(a)
```

Out[17]: 5

In [18]:
```
a='naveen sathish'
len(a)
```

Out[18]: 14

arguments in functions

In [ ]:
```
- line can call a function by using the following types
- requred arguments
- keyword arguments
```

In [1]:
```
def add(a,b):
    c=a+b
    return c
a=int(input('enter a value'))
b=int(input('enter b value'))
```

```
enter a value2
enter b value3
```

In [2]:
```
def add(a,b):
    c=a+b
    return c
a=int(input('enter a value'))
b=int(input('enter b value'))
add(a,b)
```

```
enter a value2
enter b value3
```

Out[2]:
```
5
```

In [7]:
```
#keyword arguments
def key(str):
    print(str)
key(str=123)
```

```
123
```

In [13]:
```
def keyword(name,clz):
    print('name:',name)
    print('clz:',clz)
keyword(clz='Aits',name='abc')
```

```
name: abc
clz: Aits
```

In [14]:
```
# default arguments
def default(a=10,b=1):
    print(a,b)
default(a,b)
```

```
2 3
```

In [20]:
```
def default(l,r=1):
    print(l,r)
default(l='11',r='a')
default(l='13')
```

```
1 a
1 1
```

In [5]:
```
# n odd numbers using functions
n=int(input('enter value'))
def odd(n):
    for i in range(1,n+1):
        if i%2 !=0:
            print(i,end=' ')
    return
odd(n)
```

```
enter value40
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39
```

prime or not

In [8]:
```
n=int(input('enter a value'))
def prime(n):
    c=0
    for i in range(1,n+1):
        if n%i==0:
         c=c+1
    if c==2:
        print(n,'is prime')
    else:
        print('not a prime')
prime(n)
```

```
enter a value8
not a prime
```

In [9]:
```
n=int(input('enter a value'))
def prime(n):
    c=0
    for i in range(1,n+1):
        if n%i==0:
         c=c+1
    if c==2:
        print(n,'is prime')
    else:
        print('not a prime')
prime(n)
```

```
enter a value5
5 is prime
```

In [ ]: