Amitoj Singh

03 Feb Assignment

Q1

Q1. Which keyword is used to create a function? Create a function to return a list of odd numbers in the range of 1 to 25.

In [8]:
```
# def is the keyword used to create functions
def odd_num():
    l1=[]
    for i in range(1,26):
        if i%2!=0:
            l1.append(i)
    return l1
```

In [9]:
```
odd_num()
```

Out[9]:
```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25]
```

Q2

Why **args and** *kwargs is used in some functions? Create a function each for **args and* *kwargs to demonstrate their use.

The special syntax *args in function definitions in python is used to pass a variable number of arguments to a function. It is used to pass a non-key worded, variable-length argument list. The special syntax* *kwargs in function definitions in python is used to pass a keyworded, variable-length argument list. We use the name kwargs with the double star. The reason is that the double star allows us to pass through keyword arguments (and any number of them).

In [3]:
```
#*args
def myfunc(*args):
    return "song","Thank you",[1,4,56]
```

In [4]:
```
myfunc()
```

Out[4]:
```
('song', 'Thank you', [1, 4, 56])
```

In [20]:
```
#kwargs
def newfun(**kwargs):
    return kwargs
```

In [22]:
```
newfun(a=1,c=2,d=4)
```

Out[22]:
```
{'a': 1, 'c': 2, 'd': 4}
```

Q3

What is an iterator in python? Name the method used to initialise the iterator object and the method used for iteration. Use these methods to print the first five elements of the given list [2, 4, 6, 8, 10, 12, 14, 16, 18, 20].

An iterator is an object that contains a countable number of values.An iterator is an object that can be iterated upon, meaning that you can traverse through all the values. To initialize an iterator object in Python, we use the built-in iter() function, which takes an iterable object as an argument and returns an iterator object.

In [12]:
```
ln=[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
ok_myml=iter(ln)
for i in range(5):
    print(next(ok_myml))
```

```
2
4
6
8
10
```

Q4

A generator function in Python is a special type of function that generates a sequence of values using the "yield" keyword instead of "return". When a generator function is called, it returns a generator object which can be used to iterate over the sequence of values generated by the function. Each time the "yield" keyword is encountered in the function, it suspends execution of the function and returns the yielded value to the caller. The next time the generator is called, execution resumes where it left off, allowing the function to generate the next value in the sequence.

The "yield" keyword is used in generator functions to produce a value without exiting the function. It allows the function to be paused and resumed, so that the generator can produce a sequence of values one at a time. Each time the generator is resumed, it continues executing from the point where it was last suspended, allowing it to generate the next value in the sequence.

In [13]:
```
def fibonacci(n):
    a, b = 0, 1
    for i in range(n):
        yield a
        a, b = b, a + b
```

In [14]:
```
for fib in fibonacci(10):
    print(fib)
```

```
0
1
1
2
3
5
8
13
21
34
```

Q5

Create a generator function for prime numbers less than 1000. Use the next() method to print the first 20 prime numbers

In [19]:
```
#making a list and keeping all those prime numbers in list l1
l1=[]
n=1000
for i in range(n):
    if i>1:
        for j in range(2,i):
            if i%j==0:
                break
        else:
            l1.append(i)
```

In [20]:
```
#showing the list
l1
```

```
 647,
 653,
 659,
 661,
 673,
 677,
 683,
 691,
 701,
 709,
 719,
 727,
 733,
 739,
 743,
 751,
 757,
 761,
 769,
 773,
```

In [21]:
```
#using the generator function
out=iter(l1)
for i in range(21):
    print(next(out))
```

```
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
```

Q6

Write a python program to print the first 10 Fibonacci numbers using a while loop.

In [29]:
```
def fib_Ser():
    a,b=0,1
    while True:
        yield a
        a,b=b , a+b
```

In [28]:
```
prog=fib_Ser()
```

In [30]:
```
for i in range(0,11):
    print(next(prog))
```

```
0
1
1
2
3
5
8
13
21
34
55
```

Q7

Write a List Comprehension to iterate through the given string: ‘pwskills’.

Expected output: ['p', 'w', 's', 'k', 'i', 'l', 'l', 's']

In [53]:
```
s="pwskills"
```

In [59]:
```
list(map(lambda x:x,s))
```

Out[59]:
```
['p', 'w', 's', 'k', 'i', 'l', 'l', 's']
```

Q8

Write a python program to check whether a given number is Palindrome or not using a while loop.

In [60]:
```
n=int(input("Enter number:"))
temp=n
rev=0
while(n>0):
    dig=n%10
    rev=rev*10+dig
    n=n//10
if(temp==rev):
    print("The number is a palindrome!")
else:
    print("The number isn't a palindrome!")
```

```
Enter number:121
The number is a palindrome!
```

Q9

Write a code to print odd numbers from 1 to 100 using list comprehension

In [2]:
```
numbers = []
for i in range(101):
    numbers.append(i)
```

In [3]:
```
numbers
```

Out[3]:
```
[0,
 1,
 2,
 3,
 4,
 5,
 6,
 7,
 8,
 9,
 10,
 11,
 12,
 13,
 14,
 15,
 16,
 17,
 18,
 19,
 20,
 21,
 22,
 23,
 24,
 25,
 26,
 27,
 28,
 29,
 30,
 31,
 32,
 33,
 34,
 35,
 36,
 37,
 38,
 39,
 40,
 41,
 42,
 43,
 44,
 45,
 46,
 47,
 48,
 49,
 50,
 51,
 52,
 53,
 54,
 55,
 56,
```

```
57,
58,
59,
60,
61,
62,
63,
64,
65,
66,
67,
68,
69,
70,
71,
72,
73,
74,
75,
76,
77,
78,
79,
80,
81,
82,
83,
84,
85,
86,
87,
88,
89,
90,
91,
92,
93,
94,
95,
96,
97,
98,
99,
100]
```

In [14]:
```
odd_num= []
for i in range(101):
    if i%2!=0:
        odd_num.append(i)
```

In [15]:
```
odd_num
```

Out[15]:
```
[1,
 3,
 5,
 7,
 9,
 11,
 13,
 15,
 17,
 19,
 21,
 23,
 25,
 27,
 29,
 31,
 33,
 35,
 37,
 39,
 41,
 43,
 45,
 47,
 49,
 51,
 53,
 55,
 57,
 59,
 61,
 63,
 65,
 67,
 69,
 71,
 73,
 75,
 77,
 79,
 81,
 83,
 85,
 87,
 89,
 91,
 93,
 95,
 97,
 99]
```

In [ ]: