assesment3

June 7, 2023

```
[ ]: # Q1
     "def keyword is used to create a function"
```

```
[34]: def odd_number():
          odd_number=[]
          for num in range(1,26):
              if num % 2!= 0:
                  odd_number.append(num)
          return odd_number
      print(odd_number())
```

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

```
[ ]: # Q2 :
     # the * args is used because we can use the nth  number of variable element␣
      ↪without editing any thing in a present element
     # the ** args is used because we can use the nth number of element in the form␣
      ↪of dic (key : value ) withount making change in the present value
```

```
[5]:  def function_args(*args):
         return (args)
     function_args ('sudh','chirag','shashank','element','the gamer' ,'the␣
      ↪programmer')
```

```
[5]: ('sudh', 'chirag', 'shashank', 'element', 'the gamer', 'the programmer')
```

```
[1]: def function_kwargs(**kwargs):
         return kwargs
```

```
[6]: function_kwargs(a=[1,2,34,56,] , b='chirag' , c= 23.455 ,d = True)
```

```
[6]: {'a': [1, 2, 34, 56], 'b': 'chirag', 'c': 23.455, 'd': True}
```

```
[ ]: # Q3
     " an iterator in python used to itertate the  next no of elements in a given␣
      ↪list,tuple,dic etc
```

```
'iter function is used in initial the object and the next is used for the␣
↪iteration'
```

[29]:

```
my_list = [2,4,6,8,10,12,14,16,18,20]
my_iterator = iter(my_list)
for _ in range(5):
    element = next(my_iterator)
    print(element)
```

```
2
4
6
8
10
```

[ ]:

```
#Q4 :
'generator function used to get the output for each element which is present in␣
↪variable it doesnt return all the value at once'
'yield keyword is used as it excute only one by one element which is present in␣
↪variable executes untill the condition is false'
```

[40]:

```
 def my_generator(n):
        a,b=0,1
        for iteam in range(n):
            yield (a)
            a,b=b,a+b
```

[42]:

```
for item in my_generator(100):
    print(item)
```

```
0
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
```

```
987
1597
2584
4181
6765
10946
17711
28657
46368
75025
121393
196418
317811
514229
832040
1346269
2178309
3524578
5702887
9227465
14930352
24157817
39088169
63245986
102334155
165580141
267914296
433494437
701408733
1134903170
1836311903
2971215073
4807526976
7778742049
12586269025
20365011074
32951280099
53316291173
86267571272
139583862445
225851433717
365435296162
591286729879
956722026041
1548008755920
2504730781961
4052739537881
6557470319842
```

```
10610209857723
17167680177565
27777890035288
44945570212853
72723460248141
117669030460994
190392490709135
308061521170129
498454011879264
806515533049393
1304969544928657
2111485077978050
3416454622906707
5527939700884757
8944394323791464
14472334024676221
23416728348467685
37889062373143906
61305790721611591
99194853094755497
160500643816367088
259695496911122585
420196140727489673
679891637638612258
1100087778366101931
1779979416004714189
2880067194370816120
4660046610375530309
7540113804746346429
12200160415121876738
19740274219868223167
31940434634990099905
51680708854858323072
83621143489848422977
135301852344706746049
218922995834555169026
```

[43]:

```
def my_generator1(n):
    a,b=0,1
    for iteam in range(n):
        yield (a)
        a,b=b,a+b
```

[45]:

```
for i  in my_generator(200):
    print(i)
```

```
0
1
```

```
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
10946
17711
28657
46368
75025
121393
196418
317811
514229
832040
1346269
2178309
3524578
5702887
9227465
14930352
24157817
39088169
63245986
102334155
165580141
267914296
433494437
701408733
1134903170
1836311903
2971215073
4807526976
7778742049
```

```
12586269025
20365011074
32951280099
53316291173
86267571272
139583862445
225851433717
365435296162
591286729879
956722026041
1548008755920
2504730781961
4052739537881
6557470319842
10610209857723
17167680177565
27777890035288
44945570212853
72723460248141
117669030460994
190392490709135
308061521170129
498454011879264
806515533049393
1304969544928657
2111485077978050
3416454622906707
5527939700884757
8944394323791464
14472334024676221
23416728348467685
37889062373143906
61305790721611591
99194853094755497
160500643816367088
259695496911122585
420196140727489673
679891637638612258
1100087778366101931
1779979416004714189
2880067194370816120
4660046610375530309
7540113804746346429
12200160415121876738
19740274219868223167
31940434634990099905
51680708854858323072
83621143489848422977
```

```
135301852344706746049
218922995834555169026
354224848179261915075
573147844013817084101
927372692193078999176
1500520536206896083277
2427893228399975082453
3928413764606871165730
6356306993006846248183
10284720757613717413913
16641027750620563662096
26925748508234281076009
43566776258854844738105
70492524767089125814114
114059301025943970552219
184551825793033096366333
298611126818977066918552
483162952612010163284885
781774079430987230203437
1264937032042997393488322
2046711111473984623691759
3311648143516982017180081
5358359254990966640871840
8670007398507948658051921
14028366653498915298923761
22698374052006863956975682
36726740705505779255899443
59425114757512643212875125
96151855463018422468774568
155576970220531065681649693
251728825683549488150424261
407305795904080553832073954
659034621587630041982498215
1066340417491710595814572169
1725375039079340637797070384
2791715456571051233611642553
4517090495650391871408712937
7308805952221443105020355490
11825896447871834976429068427
19134702400093278081449423917
30960598847965113057878492344
50095301248058391139327916261
81055900096023504197206408605
131151201344081895336534324866
212207101440105399533740733471
343358302784187294870275058337
555565404224292694404015791808
898923707008479989274290850145
```

```
1454489111232772683678306641953
2353412818241252672952597492098
3807901929474025356630904134051
6161314747715278029583501626149
9969216677189303386214405760200
16130531424904581415797907386349
26099748102093884802012313146549
42230279526998466217810220532898
68330027629092351019822533679447
110560307156090817237632754212345
178890334785183168257455287891792
289450641941273985495088042104137
468340976726457153752543329995929
757791618667731139247631372100066
1226132595394188293000174702095995
1983924214061919432247806074196061
3210056809456107725247980776292056
5193981023518027157495786850488117
8404037832974134882743767626780173
13598018856492162040239554477268290
22002056689466296922983322104048463
35600075545958458963222876581316753
57602132235424755886206198685365216
93202207781383214849429075266681969
150804340016807970735635273952047185
244006547798191185585064349218729154
394810887814999156320699623170776339
638817435613190341905763972389505493
1033628323428189498226463595560281832
1672445759041379840132227567949787325
2706074082469569338358691163510069157
4378519841510949178490918731459856482
7084593923980518516849609894969925639
11463113765491467695340528626429782121
18547707689471986212190138521399707760
30010821454963453907530667147829489881
48558529144435440119720805669229197641
78569350599398894027251472817058687522
127127879743834334146972278486287885163
205697230343233228174223751303346572685
332825110087067562321196029789634457848
538522340430300790495419781092981030533
871347450517368352816615810882615488381
1409869790947669143312035591975596518914
2281217241465037496128651402858212007295
3691087032412706639440686994833808526209
5972304273877744135569338397692020533504
9663391306290450775010025392525829059713
```

```
15635695580168194910579363790217849593217
2529908688864586456855589389182743678652930
40934782466626840596168752972961528246147
66233869353085486281758142155705206899077
107168651819712326877926895128666735145224
173402521172797813159685037284371942044301
```

[ ]:

```
#Q5
```

[69]:

```
def generate_prime():
    prime = []
    num  = 2
    while num<1000:
        is_prime =True
        for prime in primes:
            if num%prime==0:
                is_prime = False
                break
        if is_prime:
            prime.append(num)
            yield num
        num+=1
prime_generator = generate_primes()
for _ in range(20):
    prime_number=next(prime_generator)
    print(prime_number)
```

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

[ ]:

```
#Q6
```

[71]:

```
def fibo_nac(n):
    a,b=1,0
    for i in range(n):
        yield(a)
        a,b=b,b+a
```

[73]:

```
for i in fibo_nac(10):
    print(i)
```

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

[ ]:

```
#Q7
```

[74]:

```
s = 'pwskills'
```

[81]:

```
output = [char for char in s if char in 'pwskill']
print(output)
```

```
['p', 'w', 's', 'k', 'i', 'l', 'l', 's']
```

[ ]:

```
#Q8
```

[82]:

```
def is_palindrome(number):
    original_number = number
    reversed_number = 0
    while number > 0:
        digit = number % 10
        reversed_number = (reversed_number * 10) + digit
        number = number // 10

    if original_number == reversed_number:
        return True
    else:
        return False
num = int(input("Enter a number: "))
```

```
if is_palindrome(num):
    print(num, "is a palindrome!")
else:
    print(num, "is not a palindrome!")
```

```
Enter a number:  3
```

```
3 is a palindrome!
```

[ ]:

```
#Q9
```

[84]:

```
odd_num = [num for num in range(1,101) if num %2!=0]
for  number in odd_num:
    print(number)
```

```
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
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

```
[ ]:
```