Ass 3 (1)

August 10, 2023

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

[3]:
```
odd_numbers_list = get_odd_numbers()
print(odd_numbers_list)
```

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

[ ]:
```
[2.] *args and **kwargs are special syntax in Python that allow a function to␣
 ↪accept a variable number of arguments or keyword arguments,
respectively.
args is used to pass a variable number of non-keyword arguments to a function.␣
 ↪In the function definition, we use an asterisk (*)
before the parameter name to indicate that it should accept any number of␣
 ↪arguments. The arguments are then passed as a tuple to the
function.

**kwargs is used to accept any number of keyword arguments and print them out␣
 ↪in the format "key: value".
Both of these functions can accept any number of arguments or keyword␣
 ↪arguments, making them more flexible and versatile.
```

[1]:
```
def my_sum(*args):
    total = 0
    for num in args:
        total += num
    return total
```

[3]:
```
print(my_sum(1, 2, 3, 4, 5))
```

```
15
```

[4]:
```
def print_kwargs(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

print_kwargs(name="John", age=30, city="New York")
```

```
name: John
age: 30
city: New York
```

[5]:
```
[3.] An iterator in Python is an object that can be used to traverse the␣
↪elements of a sequence one at a time. Itertools
     module in Python provides a variety of iterators for different purposes.

    The method used to initialize the iterator object is iter(). This method␣
↪takes a sequence as its argument and returns
    an iterator object that can be used to traverse the elements of the␣
↪sequence.

    The method used for iteration is next(). This method is used to get the␣
↪next element from the iterator. If there are
    no more elements left in  the iterator, the next() method will raise a␣
↪StopIteration exception.
```

[8]:
```
def print_first_five_elements(list_of_numbers):
  """Prints the first five elements of the given list."""
  iterator = iter(list_of_numbers)
  for i in range(5):
    print(next(iterator))

list_of_numbers = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
print_first_five_elements(list_of_numbers)
```

```
2
4
6
8
10
```

[ ]:
```
[4.]  A generator function in Python is a function that returns an iterator. It␣
↪is a special type of function that does not
     return all of its  values at once, but rather yields them one at a time.␣
↪This makes generator functions memory
     efficient, as they do not need to store all of the values in memory at␣
↪once.
```

```
  The yield keyword is used in generator functions to produce a value. When␣
↪the generator function is called, it does not
  execute the  function body immediately. Instead, it returns a generator␣
↪object that can be iterated over to produce the
  values. When the next() method is called on the generator object, the yield␣
↪statement is executed and the value is
  returned. The function body then resumes execution from the next statement␣
↪after the yield statement.
```

[9]:
```
def fibonacci(n):

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

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

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

[ ]:
```
[5.] below
```

[1]:
```
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

def prime_nums_generator():
    n = 2
    while True:
        if is_prime(n):
            yield n
        n += 1
        if n >= 1000:
            break

primes = prime_nums_generator()
print("First 20 prime numbers:")
for i in range(20):
    print(next(primes))
```

```
First 20 prime numbers:
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
```

[ ]:
```
[6.] below
```

[3]:
```
first = 0
second = 1

print(first)
print(second)

count = 2
while count < 10:
    next_num = first + second
    print(next_num)

     first = second
    second = next_num
    count += 1
```

```
  Cell In[3], line 12
    first = second
    ^
IndentationError: unexpected indent
```

[5]:
```
# Program to display the Fibonacci sequence up to n-th term

nterms = int(input("How many terms? "))

# first two terms
n1, n2 = 0, 1
count = 0

# check if the number of terms is valid
if nterms <= 0:
   print("Please enter a positive integer")
# if there is only one term, return n1
elif nterms == 1:
   print("Fibonacci sequence upto",nterms,":")
   print(n1)
# generate fibonacci sequence
else:
   print("Fibonacci sequence:")
   while count < nterms:
       print(n1)
       nth = n1 + n2
       # update values
       n1 = n2
       n2 = nth
       count += 1
```

```
How many terms?  7

Fibonacci sequence:
0
1
1
2
3
5
8
```

[ ]:
```
[7.]
```

[6]:
```
string = 'pwskills'
char_list = [char for char in string]
print(char_list)
```

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

[ ]:
```
[8.]
```

[8]:
```
num = int(input("Enter a number: "))
temp = num
reverse = 0

while temp > 0:
    digit = temp % 10
    reverse = reverse * 10 + digit
    temp //= 10

if num == reverse:
    print(num, "is a palindrome")
else:
    print(num, "is not a palindrome")
```

```
Enter a number:  121
```

```
121 is a palindrome
```

[ ]:
```
[9.]
```

[ ]:
```

```