Python Practice Questions

February 16, 2022

Let's `import` useful packages for this assignment.

[1]:
```python
import functools
```

1Q) Write a function that inputs a number and prints the multiplication table of that number.

[2]:
```python
def multiplication_table(n=1, l=10) -> None:
    """
    This function prints the multiplication table of the given number.
    """
    if isinstance(n, int) and isinstance(l, int):
        if l <= 0: print('The limit should always be greater than 0.')
        else:
            for i in range(1, l+1):
                print("{}\t* {}\t= {}".format(n, i, n*i))
    else: print("Please input an integer.")
    return None
```

[3]:
```python
multiplication_table(n=563)
```

```
563     * 1     = 563
563     * 2     = 1126
563     * 3     = 1689
563     * 4     = 2252
563     * 5     = 2815
563     * 6     = 3378
563     * 7     = 3941
563     * 8     = 4504
563     * 9     = 5067
563     * 10    = 5630
```

[4]:
```python
multiplication_table(n=-3)
```

```
-3      * 1     = -3
-3      * 2     = -6
-3      * 3     = -9
-3      * 4     = -12
-3      * 5     = -15
```

```
-3      * 6     = -18
-3      * 7     = -21
-3      * 8     = -24
-3      * 9     = -27
-3      * 10    = -30
```

[5]:
```python
multiplication_table(n=5.5)
```

```
Please input an integer.
```

[6]:
```python
multiplication_table(n=25, l=0)
```

```
The limit should always be greater than 0.
```

[7]:
```python
multiplication_table(n='3')
```

```
Please input an integer.
```

2Q) Write a program to print twin primes less than 1000. If two consecutive odd numbers are both prime then they are known as twin primes.

[8]:
```python
def primality_test(n, d=2) -> bool:
    """
    This is a boolean function to know if the number is prime or not.
    """
    if isinstance(n, int):
        if n == d: return True
        elif n % d == 0: return False
        elif n <= 1: return False
        else: return primality_test(n, d=d+1)
    else: return None
```

[9]:
```python
print(primality_test(n=1))
```

```
False
```

[10]:
```python
print(primality_test(n=11))
```

```
True
```

[11]:
```python
print(primality_test(n=-2))
```

```
False
```

[12]:
```python
def find_all_primes(n=1000) -> list:
    """
    This function gets all the prime numbers till n in a list.
    """
    return [i for i in range(2, n+1) if primality_test(n=i)]
```

[13]:
```python
print(find_all_primes())
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73,
79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163,
167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251,
257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349,
353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443,
449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557,
563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647,
653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757,
761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863,
877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983,
991, 997]
```

[14]:
```python
def get_twin_primes_1(n=1000, diff=2) -> list:
    """
    This function gets all the twin primes from 0 to n.
    """
    primes = find_all_primes(n=n)
    twin_primes = list()
    i = 1
    while i < len(primes):
        if abs(primes[i-1] - primes[i]) == diff:
            twin_primes.append((primes[i-1], primes[i]))
        i += 1
    return twin_primes
```

[15]:
```python
print(get_twin_primes_1())
```

```
[(3, 5), (5, 7), (11, 13), (17, 19), (29, 31), (41, 43), (59, 61), (71, 73),
(101, 103), (107, 109), (137, 139), (149, 151), (179, 181), (191, 193), (197,
199), (227, 229), (239, 241), (269, 271), (281, 283), (311, 313), (347, 349),
(419, 421), (431, 433), (461, 463), (521, 523), (569, 571), (599, 601), (617,
619), (641, 643), (659, 661), (809, 811), (821, 823), (827, 829), (857, 859),
(881, 883)]
```

Another approach, just by understanding the question → our sample space is only limited to odd numbers.

[16]:
```python
def get_twin_primes_2(n=1000, diff=2):
    """
    This function gets all the twin primes from 0 to n.
    """
    odds = list(range(1, n, 2))
    twin_primes = list()
    i = 1
    while i < len(odds):
        if (abs(odds[i-1] - odds[i]) == diff and
            primality_test(n=odds[i-1]) and
```

```python
            primality_test(n=odds[i])):
            twin_primes.append((odds[i-1], odds[i]))
        i += 1
    return twin_primes
```

[17]:
```python
print(get_twin_primes_2())
```

```
[(3, 5), (5, 7), (11, 13), (17, 19), (29, 31), (41, 43), (59, 61), (71, 73),
(101, 103), (107, 109), (137, 139), (149, 151), (179, 181), (191, 193), (197,
199), (227, 229), (239, 241), (269, 271), (281, 283), (311, 313), (347, 349),
(419, 421), (431, 433), (461, 463), (521, 523), (569, 571), (599, 601), (617,
619), (641, 643), (659, 661), (809, 811), (821, 823), (827, 829), (857, 859),
(881, 883)]
```

3Q) Write a program to find out the prime factors of a number, for example: prime factors of $56 \to 2, 2, 2, 7$.

[18]:
```python
def get_prime_factors(n) -> list:
    """
    This function computes the prime factors for the given number.
    """
    if isinstance(n, int):
        pf = list()
        while n % 2 == 0:
            pf.append(2)
            n //= 2
        for m in range(3, n+1, 2):
            while n % m == 0:
                pf.append(m)
                n //= m
        return pf
    else: return None
```

[19]:
```python
print(get_prime_factors(n=1024))
```

```
[2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
```

[20]:
```python
print(get_prime_factors(n=10359))
```

```
[3, 3, 1151]
```

[21]:
```python
print(get_prime_factors(n=999999))
```

```
[3, 3, 3, 7, 11, 13, 37]
```

[22]:
```python
print(get_prime_factors(n=369))
```

```
[3, 3, 41]
```

[23]:
```
print(get_prime_factors(n=56))
```

```
[2, 2, 2, 7]
```

4Q) Write a program to implement these formulae of permutations and combinations.

- Number of permutations of n objects taken r at a time:

$$p(n, r) = \frac{n!}{(n-r)!}$$

- Number of combinations of n objects taken r at a time:

$$c(n, r) = \frac{n!}{r!(n-r)!} = \frac{p(n, r)}{r!}$$

[24]:
```
def fact(n) -> int:
    """
    This function returns the factorial of the given number.
    """
    if isinstance(n, int):
        if n == 0: return 1
        elif n < 0: return None
        else: return functools.reduce(lambda x, y: x * y, range(1, n+1))
    else: return None
```

[25]:
```
print(fact(n=6))
```

```
720
```

[26]:
```
print(fact(n=-6))
```

```
None
```

[27]:
```
print(fact(n=0))
```

```
1
```

[28]:
```
print(fact(n='8'))
```

```
None
```

[29]:
```
def P(n, r) -> int:
    """
    This method finds the permutations of the given numbers.
    """
    if isinstance(n, int) and isinstance(r, int):
        if r > n or n < 0 or r < 0:
            print("Please ensure r > 0, n >= r and n, r belong to I.")
        else: return fact(n=n) // fact(n=n-r)
    else: return None
```

[30]:
```
print(P(n=5, r=3))
```

```
60
```

[31]:
```
P(n=5, r=6)
```

```
Please ensure r > 0, n >= r and n, r belong to I.
```

[32]:
```
def C(n, r) -> int:
    """
    This method finds the combinations of the given numbers.
    """
    if isinstance(n, int) and isinstance(r, int):
        if r > n or n < 0 or r < 0:
            print("Please ensure r > 0, n >= r and n, r belong to I.")
        else: return fact(n=n) // (fact(n=n-r) * fact(n=r))
    else: return None
```

[33]:
```
print(C(n=5, r=3))
```

```
10
```

[34]:
```
C(n=5, r=7)
```

```
Please ensure r > 0, n >= r and n, r belong to I.
```

5Q) Write a function that converts a decimal number to binary number.

[35]:
```
def dec_to_bin(n) -> str:
    """
    This funtion converts the given decimal number to binary number.
    """
    return bin(n)[2:]
```

[36]:
```
print(dec_to_bin(n=1111))
```

```
10001010111
```

6Q) Write a function `cubesum()` that accepts an integer and returns the sum of the cubes of individual digits of that number. Use this function to make functions `PrintArmstrong()` and `isArmstrong()` to print Armstrong numbers and to find whether is an Armstrong number.

[37]:
```
def cubesum(n) -> int:
    """
    This function returns the sum of cubes of digits of the given number.
    """
    return sum(list(map(lambda i: i**3, list(map(int, str(n))))))
```

[38]:
```python
print(cubesum(n=123))
```

```
36
```

[39]:
```python
def is_armstrong(an) -> bool:
    """
    This function is a test to verify given number is an armstrong number.
    """
    return cubesum(an) == an
```

[40]:
```python
print(is_armstrong(an=123))
```

```
False
```

[41]:
```python
def print_armstrong(num) -> list:
    """
    This function lists all the armstrong numbers in [1, nums].
    """
    return list(filter(is_armstrong, range(1, num+1)))
```

[42]:
```python
print(print_armstrong(num=1000))
```

```
[1, 153, 370, 371, 407]
```

7Q) Write a function `prodDigits()` that inputs a number and returns the product of digits of that number.

[43]:
```python
def prod_digits(n) -> int:
    """
    This function returns the product of digits of the given number.
    """
    return functools.reduce(lambda x, y: x * y, list(map(int, str(n))))
```

[44]:
```python
print(prod_digits(n=45))
```

```
20
```

[45]:
```python
print(prod_digits(n=369))
```

```
162
```

8Q) If all the digits of a number n are multiplied by each other repeating with the product, the one digit number obtained at the last is called the multiplicative digital root of n. The number of times digits need to be multiplied to reach one digit is called multiplicative persistance of n.

For example:

- $86 \rightarrow 48 \rightarrow 32 \rightarrow 6$ where MDR 6 and MPersistence 3.
- $341 \rightarrow 12 \rightarrow 2$ where MDR 2 and MPersistence 2.

Using the function `prodDigits()` of previous exercise write functions `MDR()` and `MPersistence()` that input a number and return its multiplicative digital root and multiplicative persistance respectively.

[46]:
```
class MDR_MP(object):
    """
    A class for problem 8.
    """
    def __init__(self, n):
        self.n = abs(n)
        self.result = [self.n]

    def do_prod_digits(self) -> int:
        """
        This method does the product of all the digits of the number.
        """
        return functools.reduce(lambda x, y: x * y, list(map(int, str(self.n))))

    def MDR(self) -> int:
        """
        This method calculates multiplicative digital root of the number.
        """
        if len(str(self.n)) == 1:
            print("Result: {}".format(self.result))
            return self.n
        elif len(str(self.n)) > 1:
            self.n = self.do_prod_digits()
            self.result += [self.n]
            return self.MDR()

    def MP(self) -> int:
        """
        This method calculates multiplicative persistance of the number.
        Invoke this method only after invoking MDR() for proper solution.
        """
        return len(self.result[1:])

    def solution(self):
        return [self.MDR(), self.MP()]
```

[47]:
```
mdr_mp = MDR_MP(n=86)
sol = mdr_mp.solution()
print("MDR: {}".format(sol[0]))
print("MP: {}".format(sol[1]))
```

```
Result: [86, 48, 32, 6]
MDR: 6
MP: 3
```

[48]:
```
mdr_mp = MDR_MP(n=341)
sol = mdr_mp.solution()
print("MDR: {}".format(sol[0]))
print("MP: {}".format(sol[1]))
```

```
Result: [341, 12, 2]
MDR: 2
MP: 2
```

[49]:
```
mdr_mp = MDR_MP(n=36493)
sol = mdr_mp.solution()
print("MDR: {}".format(sol[0]))
print("MP: {}".format(sol[1]))
```

```
Result: [36493, 1944, 144, 16, 6]
MDR: 6
MP: 4
```

[50]:
```
mdr_mp = MDR_MP(n=5)
sol = mdr_mp.solution()
print("MDR: {}".format(sol[0]))
print("MP: {}".format(sol[1]))
```

```
Result: [5]
MDR: 5
MP: 0
```

9Q) Write a function `sumPdivisors()` that finds the sum of proper divisors of a number. Proper divisors of a number are those numbers by which the number is divisible, except the number itself. For example, proper divisors of 36 are $1, 2, 3, 4, 6, 9, 12, 18$.

[51]:
```
def sum_proper_divisors(n) -> int:
    """
    This function calculates the sum of proper divisors of a number.
    """
    return sum([i for i in range(1, n) if n % i == 0])
```

[52]:
```
print(sum_proper_divisors(n=36))
```

```
55
```

[53]:
```
print(sum_proper_divisors(n=369))
```

```
177
```

[54]:
```
print(sum_proper_divisors(n=99999))
```

```
48513
```

10Q) A number is called perfect if the sum of proper divisors of that number is equal to the number. For example 28 is perfect number, since $1 + 2 + 4 + 7 + 14 = 28$. Write a program to print all the perfect numbers in a given range.

[55]:
```
def perfect_numbers(n) -> list:
    """
    This function lists all the perfect numbers in [1, n).
    """
    return [p for p in range(1, n) if sum_proper_divisors(n=p) == p]
```

[56]:
```
print(perfect_numbers(100))
```

```
[6, 28]
```

[57]:
```
print(perfect_numbers(1000))
```

```
[6, 28, 496]
```

11Q) Two different numbers are called amicable numbers if the sum of the proper divisors of each is equal to the other number. For example 220 and 284 are amicable numbers.

- Sum of proper divisors of $220 = 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284$.
- Sum of proper divisors of $284 = 1 + 2 + 4 + 71 + 142 = 220$.
- Write a function to print pairs of amicable numbers in a range.

[58]:
```
def amicable_numbers(n1, n2) -> bool:
    """
    This function checks the amicability test of two given numbers.
    """
    return sum_proper_divisors(n=n1) == n2 and sum_proper_divisors(n=n2) == n1
```

[59]:
```
print(amicable_numbers(n1=220, n2=284))
```

```
True
```

[60]:
```
def amicable_pairs(n) -> list:
    """
    This function lists all pairs of amicable numbers.
    """
    return [(n1, n2)
            for n1 in range(1, n)
            for n2 in range(1, n)
            if amicable_numbers(n1=n1, n2=n2) and n1 != n2]
```

[61]:
```
print(amicable_pairs(n=300))
```

```
[(220, 284), (284, 220)]
```

12Q) Write a program which can filter odd numbers in a list by using `filter()`.

[62]:
```
is_odd = lambda x: x % 2 == 1
```

[63]:
```
print(list(filter(is_odd, range(1, 10))))
```

```
[1, 3, 5, 7, 9]
```

[64]:
```
print(list(filter(is_odd, range(0, 10, 2))))
```

```
[]
```

13Q) Write a program which can `map()` to make a list whose elements are cube of elements in a given list.

[65]:
```
cube_it = lambda x: x**3
```

[66]:
```
print(list(map(cube_it, range(1, 11))))
```

```
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

14Q) Write a program which can `map()` and `filter()` to make a list whose elements are cube of even number in a given list.

[67]:
```
is_even = lambda x: x % 2 == 0
```

[68]:
```
print(list(map(cube_it, filter(is_even, range(1, 11)))))
```

```
[8, 64, 216, 512, 1000]
```

End of the file.