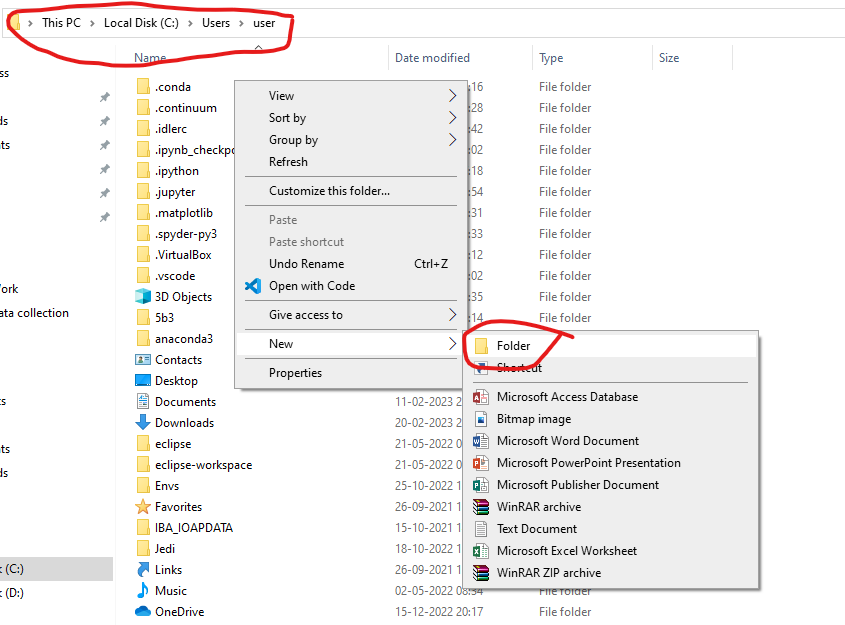
***Creation of a Library***

**Step 1:**

Create a folder in where you have to save your Library (Working Directory).

Name the folder as your Library name.

Here we named it as library.

**Step 2:**

Create \_\_init\_\_.py file in the folder to treat it as a Library/Package by the PVM.

**Step 3:**

Create a new folder for the package and name it

Here we named it as number – because we are working with number checks.

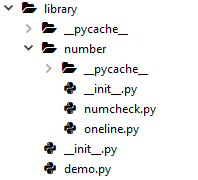
**Step 4:**

Create a \_\_init\_\_.py file in the *number* folder to it as a Package.

**Note:** Library and Package are same in nature and properties but in case of Package it is a collection of similar modules and Library is a collection of Packages.

**Step 5:**

Create modules (.py) files and save them in the in the *number* folder.

Here we created the modules called **numcheck** and **oneline**.

Here to show that even the Library can also work as a package we created a **demo** module inside the library folder.

**Description about modules:**

**numcheck :**

In the numcheck module we created the functions to check the number identities.

In this module directory of the module is

>>> import library.number.numcheck as numcheck

# We discuss about this in the comming explanation.

>>> dir(numcheck)

['\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'fact', 'isArmstrong', 'isDisarium', 'isHappy', 'isHarshad', 'isNeon', 'isPerfect', 'isPrime', 'isSpy', 'isStrong', 'isSunny', 'n\_Armstrongs', 'n\_Perfect', 'n\_Primes', 'nextArmstrong', 'nextPerfect', 'nextPrime']

In this directory \_\_.....\_\_ directory or functions are built-in functions that are automatically added by the PVM.

>>> help(numcheck)

Help on module library.number.numcheck in library.number:

NAME

library.number.numcheck - Created on Thu Dec 22 22:42:14 2022

MODULE REFERENCE

https://docs.python.org/3.10/library/library.number.numcheck.html

The following documentation is automatically generated from the Python

source files. It may be incomplete, incorrect or include features that

are considered implementation detail and may vary between Python

implementations. When in doubt, consult the module reference at the

location listed above.

DESCRIPTION

@author: 21091A05H9

FUNCTIONS

fact(n: int) -> int

Factorial : 1 \* 2 \* 3 \* 4 \* ...... \* (n-1) \* n

Parameters

----------

n : int

To find the factorial(n)

Returns

-------

int

factorial of n

isArmstrong(num: int) -> bool

Armstrong : sum of individual digits with power of number of digits in num

Parameters

----------

num : int

To check Armstrong number or not.

Returns

-------

bool

True <if n is Armstrong number>

False <if n is not a Armstrong number>

isDisarium(n: int) -> bool

Disarium : A number is a Disarium number if the sum of the digits powered with

their respective positions is equal to the number itself.

Parameters

----------

n : int

To check Disarium number or not.

Returns

-------

bool

True <if n is Disarium number>

False <if n is not a Disarium number>

isHappy(n: int) -> bool

Happy : if a number leads to 1 after a sequence of steps where in each step

number is replaced by sum of squares of its digit

Parameters

----------

n : int

To check Happy number or not.

Returns

-------

bool

True <if n is Happy number>

False <if n is not a Happy number>

isHarshad(n: int) -> bool

Harshad : if a number is divisible by the sum of its digits

Parameters

----------

n : int

To check Hashad number or not.

Returns

-------

bool

True <if n is Harshad number>

False <if n is not a Harshad number>

isNeon(num: int) -> bool

Neon : sum of individual digits of its square is equal to itself <num>

Parameters

----------

num : int

To check Neon number or not.

Returns

-------

bool

True <if n is Neon number>

False <if n is not a Neon number>

isPerfect(num: int) -> bool

Perfect number : sum of factors is equal to itself <num>

Parameters

----------

num : int

To check Perfect number or not.

Returns

-------

bool

True <if n is Perfect number>

False <if n is not a Perfect number>

isPrime(num: int) -> bool

Prime : A number is divisible by 1 and itself

Parameters

----------

num : int

To check Prime number or not.

Returns

-------

bool

True <if n is Prime number>

False <if n is not a Prime number>

isSpy(n: int) -> bool

Spy : Product of individual digits is equal to itself <n>

Parameters

----------

n : int

To check Spy number or not.

Returns

-------

bool

True <if n is Spy number>

False <if n is not a Spy number>

isStrong(n: int) -> bool

Strong : if the sum of factorial of individual digit is equal to itself <n>

Parameters

----------

n : int

To check Strong number or not.

Returns

-------

bool

True <if n is Strong number>

False <if n is not a Strong number>

isSunny(n: int) -> bool

Sunny : if 1 added to the given number, then the square root of it becomes a whole number

Parameters

----------

n : int

To check Sunny number or not.

Returns

-------

bool

True <if n is Sunny number>

False <if n is not a Sunny number>

n\_Armstrongs(n: int) -> list

Armstrong : sum of individual digits with power of number of digits in num

Parameters

----------

n : int

Returns

-------

list[l]

Returns list of 'n' Armstrong numbers from 1.

n\_Perfect(n: int) -> list

Perfect number : sum of factors is equal to itself <num>

Parameters

----------

n : int

Returns

-------

list[l]

Returns List of 'n' Perfect numbers

n\_Primes(n: int) -> list

Prime : A number is divisible by 1 and itself

Parameters

----------

n : int

Returns

-------

list[l]

Returns list of 'n' Prime numbers from 2.

nextArmstrong(num: int) -> int

Armstrong : sum of individual digits with power of number of digits in num

Parameters

----------

num : int

Returns

-------

x : int

x is the first next Prime number after the n.

nextPerfect(num: int) -> int

Perfect number : sum of factors is equal to itself <num>

Parameters

----------

num : TYPE <int>

Returns

-------

x : int

x is the first next Perfect number after the n.

nextPrime(num: int) -> int

Prime : A number is divisible by 1 and itself

Parameters

----------

num : int

Returns

-------

x : int

x is the first next Prime number after the n

FILE

c:\users\user\appdata\local\programs\python\python310\lib\library\number\numcheck.py

* *Destination of the library was not as we discussed in the staring, because we created in the Directory*

**C:\Users\user\AppData\Local\Programs\Python\Python310\Lib**

To consider it as a library from any directory in the Python IDLE.

**Code in the numcheck module**

# -\*- coding: utf-8 -\*-

"""

Created on Thu Dec 22 22:42:14 2022

@author: 21091A05H9

"""

def isPrime(num: int) -> bool:

'''

Prime : A number is divisible by 1 and itself

Parameters

----------

num : int

To check Prime number or not.

Returns

-------

bool

True <if n is Prime number>

False <if n is not a Prime number>

'''

for i in range(2,num//2+1):

if num%i == 0:

return False

return True

def nextPrime(num: int) -> int:

'''

Prime : A number is divisible by 1 and itself

Parameters

----------

num : int

Returns

-------

x : int

x is the first next Prime number after the n

'''

x=num+1

while True:

if isPrime(x):

return x

x += 1

def n\_Primes(n: int) -> list:

'''

Prime : A number is divisible by 1 and itself

Parameters

----------

n : int

Returns

-------

list[l]

Returns list of 'n' Prime numbers from 2.

'''

x=2

li = []

for i in range(n):

x=nextPrime(x)

li.append(x)

return li

def isArmstrong(num: int) -> bool:

'''

Armstrong : sum of individual digits with power of number of digits in num

Parameters

----------

num : int

To check Armstrong number or not.

Returns

-------

bool

True <if n is Armstrong number>

False <if n is not a Armstrong number>

'''

snum = str(num)

li = [int(i) \*\* len(snum) for i in snum]

return sum(li) == num

def nextArmstrong(num: int) -> int:

'''

Armstrong : sum of individual digits with power of number of digits in num

Parameters

----------

num : int

Returns

-------

x : int

x is the first next Prime number after the n.

'''

x=num+1

while True:

if isArmstrong(x):

return x

x += 1

def n\_Armstrongs(n: int) -> list:

'''

Armstrong : sum of individual digits with power of number of digits in num

Parameters

----------

n : int

Returns

-------

list[l]

Returns list of 'n' Armstrong numbers from 1.

'''

x=1

li = []

for i in range(n):

x=nextArmstrong(x)

li.append(x)

return li

def isPerfect(num: int) -> bool:

'''

Perfect number : sum of factors is equal to itself <num>

Parameters

----------

num : int

To check Perfect number or not.

Returns

-------

bool

True <if n is Perfect number>

False <if n is not a Perfect number>

'''

sum = 0

for i in range(1,num//2+1):

if num % i == 0:

sum += i

return sum == num

def nextPerfect(num: int) -> int:

'''

Perfect number : sum of factors is equal to itself <num>

Parameters

----------

num : TYPE <int>

Returns

-------

x : int

x is the first next Perfect number after the n.

'''

x=num+1

while True:

if isPerfect(x):

return x

x += 1

def n\_Perfect(n: int) -> list:

'''

Perfect number : sum of factors is equal to itself <num>

Parameters

----------

n : int

Returns

-------

list[l]

Returns List of 'n' Perfect numbers

'''

x=1

li = []

for i in range(n):

x=nextPerfect(x)

li.append(x)

return li

def fact(n: int) -> int:

'''

Factorial : 1 \* 2 \* 3 \* 4 \* ...... \* (n-1) \* n

Parameters

----------

n : int

To find the factorial(n)

Returns

-------

int

factorial of n

'''

if n<=1:

return 1

return n\*fact(n-1)

def isNeon(num: int) -> bool:

'''

Neon : sum of individual digits of its square is equal to itself <num>

Parameters

----------

num : int

To check Neon number or not.

Returns

-------

bool

True <if n is Neon number>

False <if n is not a Neon number>

'''

sq=num\*num

li = [int(i) for i in str(sq)]

return sum(li) == num

def isSpy(n: int) -> bool:

'''

Spy : Product of individual digits is equal to itself <n>

Parameters

----------

n : int

To check Spy number or not.

Returns

-------

bool

True <if n is Spy number>

False <if n is not a Spy number>

'''

li=[int(i) for i in str(n)]

pro=1

for i in li:

pro \*= i

return sum(li) == pro

def isHappy(n: int) -> bool:

'''

Happy : if a number leads to 1 after a sequence of steps where in each step

number is replaced by sum of squares of its digit

Parameters

----------

n : int

To check Happy number or not.

Returns

-------

bool

True <if n is Happy number>

False <if n is not a Happy number>

'''

while n >9:

li = [int(i) \*\* 2 for i in str(n)]

n = sum(li)

if n==1 or n == 7:

return True

return False

def isSunny(n: int) -> bool:

'''

Sunny : if 1 added to the given number, then the square root of it becomes a whole number

Parameters

----------

n : int

To check Sunny number or not.

Returns

-------

bool

True <if n is Sunny number>

False <if n is not a Sunny number>

'''

from math import sqrt

x = sqrt(n+1)

return x==int(x)

def isDisarium(n: int) -> bool:

'''

Disarium : A number is a Disarium number if the sum of the digits powered with

their respective positions is equal to the number itself.

Parameters

----------

n : int

To check Disarium number or not.

Returns

-------

bool

True <if n is Disarium number>

False <if n is not a Disarium number>

'''

i=1

Sum=0

for j in str(n):

Sum += int(j)\*\*i

i += 1

return Sum == n

def isHarshad(n: int) -> bool:

'''

Harshad : if a number is divisible by the sum of its digits

Parameters

----------

n : int

To check Hashad number or not.

Returns

-------

bool

True <if n is Harshad number>

False <if n is not a Harshad number>

'''

a=str(n)

l = [int(i) for i in a]

return n % sum(l) == 0

def isStrong(n: int) -> bool:

'''

Strong : if the sum of factorial of individual digit is equal to itself <n>

Parameters

----------

n : int

To check Strong number or not.

Returns

-------

bool

True <if n is Strong number>

False <if n is not a Strong number>

'''

a = str(n)

l = [fact(int(i)) for i in a]

return sum(l) == n

**oneline:**

In this module we created the lambda functions

>>> import library.number.oneline as oneline

>>> dir(oneline)

['\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'isAnagram', 'isAutomorphic', 'isTrimorphic']

>>> help(oneline)

Help on module library.number.oneline in library.number:

NAME

library.number.oneline - Created on Mon Feb 20 08:26:01 2023

MODULE REFERENCE

https://docs.python.org/3.10/library/library.number.oneline.html

The following documentation is automatically generated from the Python

source files. It may be incomplete, incorrect or include features that

are considered implementation detail and may vary between Python

implementations. When in doubt, consult the module reference at the

location listed above.

DESCRIPTION

@author: 21091A05H9

Automorphic : A number whose square ends in the same digits as in the number itself

Trimorphic : A number whose cube ends in the same digits as in the number itself

Anagram : letters of one string can be rearranged to form the other string

FUNCTIONS

isAnagram lambda n1, n2

isAutomorphic lambda n

isTrimorphic lambda n

FILE

c:\users\user\appdata\local\programs\python\python310\lib\library\number\oneline.py

**Code in oneline**

# -\*- coding: utf-8 -\*-

"""

Created on Mon Feb 20 08:26:01 2023

@author: 21091A05H9

Automorphic : A number whose square ends in the same digits as in the number itself

Trimorphic : A number whose cube ends in the same digits as in the number itself

Anagram : letters of one string can be rearranged to form the other string

"""

isAutomorphic = lambda n:str(n\*n).endswith(str(n))

isTrimorphic = lambda n:str(n\*\*3).endswith(str(n))

isAnagram = lambda n1,n2:set(n1) == set(n2)

Let us discuss one more module in the library folder that is **demo**

>>> import library.demo as demo

>>> dir(demo)

['\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'hi']

>>> help(demo)

Help on module library.demo in library:

NAME

library.demo - Created on Sun Feb 19 14:35:49 2023

MODULE REFERENCE

https://docs.python.org/3.10/library/library.demo.html

The following documentation is automatically generated from the Python

source files. It may be incomplete, incorrect or include features that

are considered implementation detail and may vary between Python

implementations. When in doubt, consult the module reference at the

location listed above.

DESCRIPTION

@author: 21091A05H9

FUNCTIONS

hi()

FILE

c:\users\user\appdata\local\programs\python\python310\lib\library\demo.py

**Code in demo**

# -\*- coding: utf-8 -\*-

"""

Created on Sun Feb 19 14:35:49 2023

@author: 21091A05H9

"""

def hi():

print("Hello")

**Conclusions on Library, Package and Modules**

* Module is a collection of Functions, Classes, and Variables for constant values.
* Package is a collection of Modules.
* Library is a collection of Packages.
* Library and Package both are having equal properties but different behavior and priority.
* If we create and Library, Package and Module in current working directory then we can use them in our directory only.
* If we want to use any Library, Package and Module in any directory in Python IDLE then we want to save it in

C:\Users\user\AppData\Local\Programs\Python\Python310\Lib

**OR**

C:\Users\user\AppData\Local\Programs\Python\Python310\Lib\site-packages

* The above Directory may change from Environment to Environment for Python Interpreter.