Scientific Computing with Python Lab

8th Session(Mar 19th)

Kaheon Kim : https://github.com/kaheonkim/Scientific-computing-with-python-lab-material

Today

we are going to talk about

- Dictionary : Problem 1~5
- Raise Error : Problem 1
- try/except : Problem 3
- cmath package : Problem 1
- Memoization : Problem 4,5

Dictionary

Basics

dictionary[a] = b

dictionary[c] = d

dictionary[e] = f

- What is the dictionary?

  dictionary = {a : b, c : d, e : f} → a,b,c,d,e,f would be any type

  - key : a, c, e

  - value : b, d, f

- Why dictionary, compared to list?

  They have wider range for the input

  We can make pairs with different types of data in a single data structure.

- Operation

  addition : dictionary[new_index] = new_item

  update : dictionary[existing_index] = new_item

Practice

2. Let $n > 1$ be a positive integer, and assume n has the prime factorization:

$$n = 2^{\alpha} p_1^{e_1} \cdots p_m^{e_m} q_1^{f_1} \cdots q_n^{f_n}$$

where the p_i are primes of the form $4k+1$ for $k \in \mathbb{Z}$ and the q_j are primes of the form $4k+3$ for $k \in \mathbb{Z}$ and all of the exponents are non-negative integers. A fact from number theory asserts that the number of ways to write n as a sum of two integer squares (distinguishing signs and order) is given by:

$$r(n) = \begin{cases} 0, & \text{if any } f_i \text{ is odd} \\ 4\prod_{i=1}^{m}(1+e_i), & \text{if all } f_i \text{ are even} \end{cases}$$

Write a function named r that accepts a dictionary representing the prime factorization of a positive integer $n > 1$ and returns the number of ways that n may be written as a sum of integer squares. Each key in the dictionary will be a distinct prime factor of n, and the corresponding value will be the power to which that prime factor appears in the prime factorization.

For example, for the integer $n = 45090045000$, the prime factorization is given by $(2^3)(3^2)(5^4)(7^2)(11^2)(13^2)$. The dictionary passed in for this n would be $\{2:3, 3:2, 5:4, 7:2, 11:2, 13:2\}$. The primes of the form $4k+3$ in this factorization are 3, 7 and 11, and all of these appear to an even power, so $r(n)$ is non-zero. We compute $r(n)$ using the exponents of the primes 5 and 13 of the form $4k+1$ as: $r(n) = 4 * (4+1) * (2+1) = 60$ (so there are 60 ways of writing 45090045000 as the sum of two integer squares).

As another example, consider the integer $n = 20037600$, the prime factorization is given by $(2^5)(3^2)(5^2)(11^2)(23^1)$. The dictionary passed in for this n would be $\{2:5, 3:2, 5:2, 11:2, 23:1\}$. The primes of the form $4k+3$ in this factorization are 3, 11 and 23. 23 appears to an odd power, so it is not possible to write n as the sum of two integer squares, and hence $r(n) = 0$ for this value of n (so it is impossible to write 20037600 as the sum of two integer squares).

Raise Error

- Errors : encourage you to put the proper input for the next time

  - ValueError : when the input is of inappropriate value

  - TypeError : when the operation is applied to nonproper type of variable

  - NameError : when the variable is not defined

  - Etc : ZeroDivisonError

- How to impose the error type?

  - raise TypeError(“Message”)

  - raise ValueError(“Message”)

  - raise NameError(“Message”)

Practice

Try/Except

To avoid the error

- Intention : If we face certain types of error, we are going to pass and continue to appropriate value in the while loop
- try:

    operations
- except Error_type

    pass

Practice

Complex Numbers

Basic format

- form

complex_number : c = a + bj

(if you want to represent $1i$, not ‘j’ but ‘1j’) : c == c.real + c.imag*1j

- Manipulation

 - Access : c.real would be a / c.imag = b

 - absolute value : abs(c) wpould be $\sqrt{a^2 + b^2}$

- cmath package

 - cmath.exp : taking exponential function with the complex input

 - cmath.phase : finding for the complex value θ

Practice

Memoization

Substitute for the Recursion

- We take a memo about the information of pairs of input and the output(key, item)
- No need to trace back to the past values by storing the information

→ Saving a lot of time cost

Scheme

1. Start with basic dictionary with given values
2. Calculate the value p(k) by using the previous values stored in the dictionary
3. Update the new value (k, p(k))

Practice

2. Write a Python script that consists of a single function definition: a function named produce_bell that accepts a single integer parameter n and returns a list of the Bell numbers B_i for $i = 0, \ldots, n$ (up to n inclusive). The Bell numbers are computed according to the following rule:

$$B_n = \begin{cases} 1 & \text{if } n = 0 \text{ or } 1 \\ \sum_{k=0}^{n-1} \binom{n-1}{k} B_k & \text{for } n \geq 2 \end{cases}$$

Note that while the definition of the Bell numbers is recursive in nature, the computation should be performed iteratively (similarly to computing the list of the Fibonacci numbers up to index n inclusive from class).

The Bell numbers are a famous sequence in combinatorics. B_n represents the number of distinct partitions of a set with n distinct elements. The first few Bell numbers are: 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975.