Python 3.x

Shalini Mittal

What we will cover today?

Assignment Solution

Data Structures

LOOP– ASSIGNMENT SOLUTION

FACTORS OF NUMBER

```
no = int(input('Enter a no to calculate factor'))
for i in range(1, no+1):
    if no % i == 0:
        print(i)
```

PRINT 1ST CHARACTER

```
str = input('Enter a string')
print(str[0],end='.')
for s in range(len(str)):
    if str[s] == ' ':
        print(str[s+1],end = '.')
```

ARMSTRONG NUMBER

```
no = int(input('Enter  a no to find if its armstrong'))
t = no
sum = 0
while t!= 0:
   r = t % 10
   sum = sum + r*r*r
   t = t//10
if sum == no:
   print("Armstrong")
else:
   print("Not armstrong")
```

Loop- ASSIGNMENT SOLUTION

```
ASCENDING ORDER
no = int(input('Enter  a number'))
t = no
for i in range(0,10):
    while no != 0:
            r = no%10
            if i==r:
                print(r)
            no = no//10
    no = t
```

Loop– Assignment Solution

```
choice = int(input('Enter  1. Palindrome\n2.Perfect'))
if choice == 1:
        no = int(input('enter  a no to find if it is palindrome'))
        orig , rev= no , 0
        while no != 0:
                r = no%10
                rev = rev * 10 + r
                no = no // 10
        if orig == rev:       print('palindrome')
        else:       print('not palindrome')
elif choice == 2:
        sum = 0
        no = int(input('enter  a no to find if it is perfect'))
        for i in range(1,no):
                if no % i ==0:
                        sum = sum + i
        if no == sum:
                print('perfect')
        else:
                print('not perfect')
```

Data Structures

Tuples

List

Dictionaries

Sets

Tuples

- A tuple is an immutable list
- A tuple is defined analogously to lists, except that the set of elements is enclosed in parentheses instead of square brackets.
- The rules for indices are the same as for lists. Once a tuple has been created, you can't add elements to a tuple or remove elements from a tuple.

- Where is the benefit of tuples?
- Tuples are faster than lists.
- If you know that some data doesn't have to be changed, you should use tuples instead of lists, because this protect your data against accidental changes to these data.
- Tuples can be used as keys in dictionaries, while lists can't.

Create Tuples

```
# empty tuple
          my_tuple = ()
# tuple having integers
          my_tuple = (1, 2, 3)
# tuple with mixed datatypes
          my_tuple = (1, "Hello", 3.4)
# nested tuple
          my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
# tuple can be created without parentheses  also called tuple packing
          my_tuple = 3, 4.6, "dog"
# tuple unpacking is also possible
          a, b, c = my_tuple
```

Indexing Tuples

```
my_tuple = ['p','e','r','m','i','t']
my_tuple[0] 'p‘
my_tuple[5] 't‘
my_tuple[6] # index must be in range
my_tuple[2.0] # index must be an integer
n_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
n_tuple[0][3] # nested index 's'
n_tuple[1][1] # nested index 4
n_tuple[2][0] # nested index 1
```

Negative Indexing

Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.

```
my_tuple = ['p','e','r','m','i','t']
my_tuple[-1] 't‘
my_tuple[-6] 'p'
```

Slicing Tuples

```
my_tuple = ('p','r','o','g','r','a','m','i','z')
my_tuple[1:4] # elements 2nd to 4th ('r', 'o', 'g')
my_tuple[:-7] # elements beginning to 2nd ('p', 'r')
my_tuple[7:] # elements 8th to end ('i', 'z')
my_tuple[:] # elements beginning to end
```

Changing or Deleting Tuples

- Tuples are immutable.
- But if the element is itself a mutable datatype like list, its nested items can be changed.
- my_tuple = (4, 2, 3, [6, 5])
- my_tuple[1] = 9 # we cannot change an element
- del my_tuple[3] # can't delete items

Tuple Methods

Method	Description
count(x)	Return the number of items that is equal to x
index(x)	Return index of first item that is equal tox
all()	Return True if all elements of the tuple are true (or if the tuple is empty).
any()	Return True if any element of the tuple is true. If the tuple is empty, return False.
enumerate()	Return an enumerate object. It contains the index and value of all the items of tuple as pairs.
len()	Return the length (the number of items) in the tuple.
max()	Return the largest item in the tuple.
min()	Return the smallest item in the tuple
sorted()	Take elements in the tuple and return a new sorted list (does not sort the tuple itself).
sum()	Retrun the sum of all elements in the tuple.
tuple()	Convert an iterable (list, string, set, dictionary) to a tuple.

List

A list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.
It can have any number of items and they may be of different types (integer, float, string etc.).
A list can even have another list as an item. These are called nested list.
# empty list my_list = []
# list of integers my_list = [1, 2, 3]
# list with mixed datatypes my_list = [1, "Hello", 3.4]
# nested list my_list = ["mouse", [8, 4, 6]]

Slicing, Indexing, Changing and deleting are same as tuples.

List methods

Method	Description
append(x)	Add item x at the end of the list
extend(L)	Add all items in given list L to the end
insert(i,x)	Insert item x at position i
remove(x)	Remove first item that is equal to x, from the list
pop([i])	Remove and return item at position i (last item if i is not provided)
clear()	Remove all items and empty the list
index(x)	Return index of first item that is equal to x
count(x)	Return the number of items that is equal to$x$
sort()	Sort items in a list in ascending order
reverse()	Reverse the order of items in a list
copy()	Return a shallow copy of the list

These methods are accessed as list.method().

Built in functions are same as tuples

SET

- Set is an unordered collection of items.
- Every element is unique (no duplicates) and must be immutable.
- However, the set itself is mutable (we can add or remove items).
- Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.
- Curly braces or the set() function can be used to create sets.
- Note: to create an empty set you have to use set(), not {}; the latter creates an empty dictionary

Create Set

- # set of integers

  my_set = {1, 2, 3}
- # set of mixed datatypes

  my_set = {1.0, "Hello", (1, 2, 3)}
- # set do not have duplicates

  {1,2,3,4,3,2} {1, 2, 3, 4}
- # set cannot have mutable items

  my_set = {1, 2, [3, 4]}
- # but we can make set from a list

  set([1,2,3,2]) {1, 2, 3}

Changing or deleting Set

- Set does not support indexing or slicing as it is unordered
- We can add single elements using the method add().
- Multiple elements can be added using update() method.
- The update() method can take tuples, lists, strings or other sets as its argument.
- In all cases, duplicates are avoided.
- my_set = {1,3}
- my_set[0] #'set' object does not support indexing
- my_set.add(2)
- my_set.update([2,3,4])
- my_set.update([4,5], {1,6,8})
- Removing Elements from a Set
- my_set.discard(4)
- my_set.remove(6)

Mathematical Operations On Set

- Union
  - Union of *A* and *B* is a set of all elements from both sets. Union is performed using | operator. Same can be accomplished using the method union().
- Intersection
  - Intersection of *A* and *B* is a set of elements that are common in both sets. Intersection is performed using & operator. Same can be accomplished using the method intersection().
- Difference
  - Difference of *A* and *B* (*A* - *B*) is a set of elements that are only in *A* but not in *B*. Similarly, *B* - *A* is a set of element in *B* but not in *A*. Difference is performed using - operator. Same can be accomplished using the method difference().
- Symmetric Difference
  - Symmetric Difference of *A* and *B* is a set of element in both *A* and *B* except those common in both. Symmetric difference is performed using ^ operator. Same can be accomplished using the methodsymmetric_difference().

Method	Description
add()	Add an element to a set
clear()	Remove all elemets form a set
copy()	Return a shallow copy of a set
difference()	Return the difference of two or more sets as a new set
difference_update()	Remove all elements of another set from this set
discard()	Remove element from set if it is a member..Do nothing if element not in set
intersection()	Return the intersection of two sets as a new set
intersection_update()	Update the set with the intersection of itself and another
isdisjoint()	Return True if two sets have a null intersection
issubset()	Return True if another set contains this set
issuperset()	Return True if this set contains another set
pop()	Remove and return an arbitary set element. Raise KeyError if the set empty
remove()	Remove an element from a set. It element inot a member, raise aKeyError
symmetric_difference()	Return the symmetric difference of two sets as a new set
symmetric_difference_update()	Update a set with the symmetric difference of itself and another
union()	Return the union of sets in a new set
update()	Update a set with the union of itself and others

Dictionary

- Unordered collection of items.
- Has a key: value pair.
- Indexed by *keys* - immutable type; strings and numbers.
- Tuples can be used as keys if they contain only strings, numbers, or tuples; if a tuple contains any mutable object either directly or indirectly, it cannot be used as a key.
- Lists cant be used as keys.
- Optimized to retrieve values when the key is known.
- Main operations are
  - Storing a value with some key and extracting the value given the key.
  - Delete a key:value pair with del.
  - Store using a key already in use, the old value associated with that key is forgotten.
- It is an error to extract a value using a non-existent key.
- list(d.keys()) - returns a list of all the keys in arbitrary order (if you want it sorted, just use sorted(d.keys()) instead).
- Check if a single key is in the dictionary, use the in keyword.

Create Dictionary

- # empty dictionary

  my_dict = {}

- # dictionary with integer keys

  my_dict = {1: 'apple', 2: 'ball'}

- # dictionary with mixed keys

  my_dict = {'name': 'John', 1: [2, 4, 3]}

- # using dict()

  my_dict = dict({1:'apple', 2:'ball'})

- # from sequence having each item as a pair

  my_dict = dict([(1,'apple'), (2,'ball')])

Access Elements

- my_dict = {'name':'Ranjit', 'age': 26}

  my_dict['name']

  my_dict.get('age') 26

Changing/Adding Elements in a Dictionary

- We can add new items or change the value of existing items using assignment operator.
- If the key is already present, value gets updated, else a new key: value pair is added to the dictionary.

```
my_dict {'age': 26, 'name': 'Ranjit'}
my_dict['age'] = 27 # update value
my_dict {'age': 27, 'name': 'Ranjit'}
 my_dict['address'] = 'Downtown' # add item
```

Deleting/removing Elements

- pop() removes as item with the provided key and returns the value.
- popitem() removes and returns an arbitrary item (key, value)
- All the items can be removed at once using the clear() method.
- We can also use the del keyword to remove individual items or the entire dictionary itself.

Method	Description
clear()	Remove all items form the dictionary.
copy()	Return a shallow copy of the dictionary.
fromkeys(*seq*[,*v*])	Return a new dictionary with keys from*seq* and value equal to *v* (defaults toNone).
get(*key*[,*d*])	Return the value of *key*. If *key* doesnot exit, return *d* (defaults to None).
items()	Return a new view of the dictionary's items (key, value).
keys()	Return a new view of the dictionary's keys.
pop(*key*[,*d*])	Remove the item with *key* and return its value or *d* if *key* is not found. If *d* is not provided and *key* is not found, raisesKeyError.
popitem()	Remove and return an arbitary item (key, value). Raises KeyError if the dictionary is empty.
setdefault(*key*[,*d*])	If *key* is in the dictionary, return its value. If not, insert *key* with a value of *d*and return *d* (defaults to None).
update([*other*])	Update the dictionary with the key/value pairs from *other*, overwriting existing keys.
values()	Return a new view of the dictionary's values

Assignments

- With a given integral number n, write a program to generate a dictionary that contains (i, i*i) such that is an integral number between 1 and n (both included). and then the program should print the dictionary.
  Suppose the following input is supplied to the program:
  8
  Then, the output should be:
  {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64}
- Create a list of dictionary to store name, email and phoneno for 5 employees.
- Create a list of palindrome words. Print the word with the longest string length along with the no of characters in the longest word.
  EX : [“mom”, “dad”,”civic”,”racecar”,”madam”, “noon”,”wow”]
  Output: racecar is longest word with 7 characters

  If possible extend this assignment to count the length of all the words in the list and print the output as follows:

```
Length  count
3       3
5       2
7       1
4       1
```

Any Question ?

Thank you !