# Other Structured Types
# Sets, Tuples, Dictionaries

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
**DELHI**

# Recap

- Variables point to objects of different types
- Objects can be of scalar types: int, float, boolean
- Objects can be structured/ compound types; built-in are: lists, strings, sets, tuples, dictionaries
- In structured types, you can perform operations on the whole object, but can also extract items from it
- So far we have discussed lists and strings
- Now we will discuss the remaining ones: tuples, sets, dictionaries
- Dictionaries in particular are very commonly used, and like strings, are a strength of python

# Recap – Lists

- Lists are a list of items in brackets, eg. [1, 4, 9, "str", 5.0, 4]
- Lists are mutable, can change the items like L[index] = val
- Can slice a list to get a sublist - from start or from end of list
- Joining or repeating lists by operations: + , *
- Functions with list as parameter: len(), sum()
- Presence/absence of item by ops: in, not in
- Ops on a list: append(), insert(), extend(), remove(), pop(), index(), reverse(), count(), copy(), sort()
- Can easily loop over list items - item by item, or using index
- Lists can be nested - i.e. list items are themselves lists
- List comprehension a compact way to create lists from lists

# Recap – Strings

- Strings are like "Hello hi" or 'Hi Hello' ; can loop over chars in str
- They are immutable, cannot change any item of a string
- Can slice a string to get substrings - from start or from end
- Functions: len(), in, not in, + , *
- Can split strings into a list of items using s.split()
- Can join a list of strings to form one using join()
- String operations (return a new string): lower(), upper(), replace(), count(), find(), isdigit(), …

# Dictionaries

# Feedback – it is anonymous

How much help you took from friends in answering quiz questions:

Green: I answered all questions without help from any friend

Yellow: I took help from friends in a few questions

Red: I took help from friends in most of the questions

# Feedback – it is anonymous

How much help you took from internet in answering quiz questions:

Green: I answered all questions without searching on the internet

Yellow: I got help from the internet in a few questions

Red: I got help from the internet in most of the questions

# Dictionary

- Dictionaries are used to store data items in key:value pairs
- key s in a dict must be unique (no duplicates)
- Values in items can be of any data type, and can be changed
- Dictionary items can be referred to by using the key name (no index)
- Dictionary is mutable but the keys must be of an immutable type.
- Duplicate items (i.e. with same key value) not permitted
- A dictionary can be created using the following syntax :

```
car = { "make": "Honda", "year": 2021,
        "model": "City","cc":1500, "price": 19.5,}
```

# Accessing Dictionary Items

- Can access any value using key and can also assign a new value. Eg: car["make"] is "Honda", car["cc"] is 1500, car["price"] is 19.5
- Can use get method also, e.g. car.get("model") will return "City"
- If we try to access a key value which is not present - KeyError.
- However, get() returns None, if key not present; we can also specify a value that is returned when the key is absent

  car.get(“Fuel”) # returns None

  car.get(“Fuel”,”Petrol”) # returns Petrol

# Keys, Values, Items

- Can get all the keys or values of a dictionary as a list

  k = car.keys() # returns all the keys

  vals = car.values() # returns all the values
- Can also get all items - this will be a list of tuples

  car.items() # returns list of type: (key, value)
- Can check whether a key exists in dictionary using the **in** keyword.

  “make” in car # Returns True

  “fuel” in car  # Returns False
- **not in** for checking absence of a key

  “fuel” not in car # Returns True
- Number of key-value pairs can be obtained using the *len()* function.

# Modifying Dictionary

- Change an item's value - just access it and assign new value

```
car["make"] = "Suzuki"
```

- Adding an item - like change, if key doesn't exist, new item created

```
car["boot"] = 450 # will add this item ("boot": 450)
```

- Removing an item - pop("key") will remove the item

```
car.pop("make") # removes the make item, returns the value
popitem() # removes the last item added, returns item
clear() # clears the dictionary
del removes the specified key from the dictionary.
```

# Looping Over a Dictionary

```
for i in car: # looping over key values
    print(i)  # print the ith key
    print(car[i]) # print the value of ith item

for i in car.values(): # looping over values
    print(val) # prints the ith item value

for i in car.items(): # each i is a tuple giving the ith item
    print(i)

for x,y in car.items():  # get the key in x and value in y
    print(x,y)
```

# Create, Copy a Dictionary

- Like lists, assigning the dictionary variable to another does not make another copy of the dictionary
- If d1 is a dictionary, d2 = d1 means that d2 points to the same dictionary as d1 - making change in one will be reflected in other.
- To make a copy, like in list, use copy method
- d2 = d1.copy() # now d2 points to a different object
- The function dict() can also be used
- A new dictionary can be created from a list of keys, with a default value for each item

  d.fromkeys(<keylist>, value)

# Nested Dictionary

- The value in each item can be a dictionary (or any structured type)
- With values being dictionaries, we have nested dictionary
- Nesting can be arbitrarily deep - giving power to represent a wide range of data
- Nested dictionaries used widely through JSON format for exchanging data

# Example of Nested Dictionary

- Let's take the record of a person at IIIT-D.

```
p1 = {"name": "Ayush", "role": "Student",
      "DOB": {"date": 4, "month": 2, "year": 2001},
      "Address": {"Colony": "Vasant Kunj", "No": 201},
           "Hobbies":    ["reading",    "movies",    "cricket"]
                                                              }
```

- p1["name"] is "Ayush"
- p1[“DOB”] is {"date": 4, "month": 2, "year": 2001}
- p1["DOB"]["year"] is 2001

# Dictionary Methods

- Various methods of a dictionary, which we have already seen

  **d.get(<key>)** # returns the value of the item with <key>

  **d.keys()** # list of keys

  **d.values()** # list of values

  **d.items()** # list of types of (key, value)

  **d.pop(<key>)** # removes the item with <key>

  **d.clear()** # clears

  **d.copy()** # copies the dictionary

  **len(d)** # size of dictionary i.e. number of keys

# Example

Count frequency of list elements

Input: [1,2,1,1,2,4,6,4,1,7]

Output:

1: 4

2: 2

4: 2

6: 1

7: 1

```
lst = [1,2,1,1,2,4,6,4,1,7]
freq = {}
for x in lst:
  if x not in freq:
    freq[x] = 0
  freq[x] = freq[x]+1
print(freq)

# Alternative Method: using get()

lst = [1,2,1,1,2,4,6,4,1,7]
freq = {}
for x in lst:
  freq[x] = freq.get(x,0)+1
print(freq)
```

# Quiz – Single Correct

What would be the output of the code given below?

```
p = {5 : 5, 7 : '7', '5' : 5, '7' : 8}
p[7] = 5
p['5'] = 7

print(p[str(p[p[str(p[5])]])])
```

a.) Error

b.) 5

c.) 7

d.) ‘7’

# Quiz – Single Correct

What would be the output of the code given below?

```
p = {5 : 5, 7 : '7', '5' : 5, '7' : 8}
p[7] = 5
p['5'] = 7

print(p[str(p[p[str(p[5])]])])
```

a.) Error

b.) 5

**c.) 7**

d.) '7'

Explanation : After update, p = {5: 5, '5': 7, 7: 5, '7': 8}

p[5] = 5

p[str(p[5]) = p('5') = 7

p[p[str(p[5])]] = p[7] = 5

p[str(p[p[str(p[5])]])] = p['5'] = 7 (Ans.)

# Example

Consider a student record in a college over different semesters

```
student = {
   "rollno": "1234",
   "name": "Shyam",
   "sem1": [("m101", 4, 9), ("cs101", 4, 8), ("com101", 4, 10)],
   "sem2": [("m102", 4, 8), ("cs102", 4, 9), ("ee102", 4, 8)],
   "sem3": [("m202", 2, 10), ("cs201", 4, 8), ("elect1", 4, 10)],
   }
```

Let us compute the SGPA for a semester

# An Example

```
def sgpa(student, sem):
    tot1 = 0
    tot2 = 0
    for i in student[sem]:
        tot1 += i[1]
        tot2 += i[1]*i[2]
    SGPA = tot2/tot1
    return SGPA

SGPA = sgpa(student, "sem1")
print(f'sgpa is {SGPA}')
```

Computing SGPA for all the semesters

```
for i in student.items():
    if i[0][:3] == "sem":
        SGPA = sgpa(student, i[0])
        print(f'sgpa for {i[0]} is {SGPA}')
```

# Summary – Dictionaries

- Dictionaries store data in key-value pairs; keys or values can be of any type
- Items in dictionary are accessed using the key as "index"
- Dictionaries are mutable, but Keys cannot change, and keys must be unique
- keys(), values(), items() - provide as lists of items/tuple
- Can add an item (by just dict[key] = value), delete by pop()
- Many methods on dictionaries
- Can loop over dictionary - using keys. Can also loop over values by getting values(), or items()

# Structured Types
# Some Commonalities

# Creating Objects using Constructor

- Python provides constructor functions to create an object of a type from other objects,

  **list(), set(), dict(), tuple(), str() # the arg has to be suitable**

- Example uses of these - valid and invalid

  T = tuple("Hello") # ('H', 'e', 'l', 'l', 'o')

  S = str(5.0) # Converted number to string. Now S = ‘5.0’

  L1 = list(T) # ['H', 'e', 'l', 'l', 'o']

  L2 = list(“Hello”)

  D = dict(a=1,b=2)  # {'a': 1, 'b': 2}

# Common Functions on Objects

There are some common functions which are useful - some apply to structured types, some to all objects

- type(x) # returns the type of x
- len(x) # returns the no of items in x
- all(x) # returns True if all elements of an iterable are true
    - all([1,1,1]) -> True ; all([1,0,0]) -> False
- any(x) # returns True if any elements of an iterable are true
    - any([1,0,0]) -> True ; any([0,0,0]) -> False
- reversed(x) # returns a reverse iterator
- sorted(x) # returns a new sorted list from the items in iterable

# enumerate() – convenient way to iterate

This is a special function that allows you to simultaneously get the indexes and values of an iterable object.

Examples:

```
l = ["Sam", "John", "Tim"]
for index, val in enumerate(l):
    print(index, val)
```

```
0 Sam
1 John
2 Tim
```

```
d = {"Sam":10, "John":30, "Tim":20}
for index, (key, val) in enumerate(d.items()):
    print(index, key, val)
```

```
0 Sam 10
1 John 30
2 Tim 20
```

# Mutable and Immutable Objects

- A variable (in python) points to an object with a value/state
- Objects are of two types: Mutable and Immutable
- Mutable objects: The state/value can be changed
- Immutable objects - state cannot be changed (though new objects can be created)
- Immutable Objects : objects of type: int, float, bool, string
- Mutable Objects: of type list, dictionary, set, (and custom objects)
- Tuple - a special case. While tuple is immutable, it may have elements (e.g. list) which are mutable

# Mutable and Immutable Objects

Immutable objects - assigning a var pointing to one to another

X = 10

Y = X

Y = 20 # a new object 20 is created to which y points, x continues to point at object with value 10

Mutable objects - assigning only creates a pointer

L1 = [1, 3, 5, 9]

L2 = L1

l1 [2] = 8 # this changes the list, and so l2 will also reflect it