Date - 30.11.23

Dictionary :-

→ Every element is a key: value pair
→ 'key' values are immutable.

1) A dictionary is an unordered sequence of key-value pairs.
2) Indices in a dictionary can be of any immutable type and are called keys.
3) Every element value corresponds to a key value.
4) Keys must be unique, but values may be duplicate.
5) Keys are immutable.

element 1 element 2 element 3

```
>>> month = {1 : 'Jan', 2 : 'Feb', 3 : 'March'}
```

key value

```
>>> month[1]  → key
Jan

>>> month.keys()
[1, 2, 3]
>>> month.values()
['Jan', 'Feb', 'Mar']
>>> month.items()
[(1, 'Jan'), (2, 'Feb'), (3, 'Mar')]
```

Pubuilt Methods for Dictionary :-

1) len()
```
>>> len(mouth)
    3
```

2) min()
```
>>> min(mouth)
    Feb
```

3) max()
```
>>> max(mouth)
    Mar
```

4) del()
```
>>> del(mouth[1])
    {2:'Feb', 3:'Mar'}.
>>> del(mouth)
```

5) get() - This method is used to extract the value corresponding to the given key.
```
>>> mouth.get(1, -1)
    Jan
```
→ user-defined
key, default (None) → if element not present

6) update()
```
>>> mouth.update(1, 'DEC')
```
→ key

This method is used to insert a modified value onto the dictionary.

7) copy()

>>> month.1.copy(month)

↓ duplicate of month

Q. WAP to merge two lists.

```
l1 = [1, 2, 3]
l2 = ['a', 'b', 'c']
zip(l1, l2)
# l3 = [(1,'a'), (2,'b'), (3,'c')] ⇒ list of tuples
```

```
l3 = l1 + l2
print(l3)
```

Date- 21.12.23

Chapter - 8 (Recursion)

* Linear Recursion :-

① $n! = n * (n-1) * (n-2) * \ldots\ldots 1$

1) def fact(n):
    fact' = 1
    for i in range(1, n+1):
        fact' = fact' * i
    return (fact')

2) def fact(n):

    if n == 0 or n == 1:
        return 1

    else:
        return n * fact(n-1)

② x^n

$$f(x^n) = \begin{cases} 1, & n = 0 \\ x * f(x, n-1), & n > 0 \end{cases}$$

def power(x, n):
    if n == 0:
        return 1
    else:
        return x * power(x, n-1)

③ Write a recursive func^n to convert decimal to binary format

Soln -

```
def binary(num):
    bin = 0
    if num == 0:
        print('0')
    elif num == 1:
        print('1')
    else:
        while(num > 0):
            bin = num % 2
            binary(num//2)
            print(bin, end = "")
```

$$f(num) = \begin{cases} 0, & num = 0 \\ 1, & num = 1 \\ f(num//2) + if\ n > 1 \end{cases}$$

$$f(n) = \begin{cases} 0, & if\ n = 0 \\ 1, & if\ n = 1 \\ f(n//2), & if\ n > 1 \end{cases}$$

```
def dtb(n):
    if n == 0:
        print('0')
    elif n == 1:
        return 1
    else:
```

```
def bin(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return (n % 2 + 10 * bin(n//2))
```

```
r = n % 2
dtb(n//2)
print(r, end = " ")
```

* Binary Recursion :-

(4) Write a recursive function to compute the n th term of fibonacci series.

$$f(n) = \begin{cases} 0, & n = 1 \\ 1, & n = 2 \\ f(n-1) + f(n-2), & n > 2 \end{cases}$$

(f(n) ↓ terms)

```
def fibo(n):
    if n == 1:
        return 0
    elif n == 2:
        return 1
    else:
        return fibo(n-1) + fibo(n-2)
```

Date - 23.12.23

1) Write a recursive program to compute length of the input string.

Soln -

$$f(str) = \begin{cases} 0, & \text{if } str = "" \\ 1 + f(str-1), & \text{if } str != "" \end{cases}$$

```
def stringlen(Str):
    if Str == "":
        return 0
```

```
    else:
        return (1 + stringlen(str[1:]))
```

2) Write a recursive program to reverse the input string.

```
def reverse(n):
    if n == "":
        return ""
    else:
        return reverse(n[1:]) + n[0]
```

3) Write a recursive program to check whether the input string is a palindrome or not.

```
def palindrome(str):
    if str[0] != str[-1]:
        return False
    else:
        return palindrome(str[1:-1]) and
        True
```

OR

```
def is-pal(str):
    if str == "":
        return True
    else:
        return str[0] == str[-1] and is-pal
            (str[1:-1])
```

4) Write a recursive function to compute total no. of vowels present in the input string.

```
def vowels (str):
    if str == "":
        return 0
    else:
        c = str[0].lower()
        if c in "aeiou":
            return 1 + vowels(str[1:])
        else:
            return 0 + vowels(str[1:])
```

Date - 04.01.2024

* File - open, create, read, write, close
copy from one to another, count no. of
character/words in a file

Ch-9 (Exception Handling)

Pg No - 227 - 238

↓
a kind of runtime error

* Exception :-

→ Exception is a kind of runtime error.

→ It terminates the program abnormally

→ In python, we have the following kinds of exception that may occur at the runtime :-

① NameError - This exception may occur whenever a name that appears in a statement is not found globally.

eg-
```
mark = Input ("Enter your mark: ")
NameError : name 'Input' is not defined
```

② TypeError - This exception occurs when an operation or function is applied on an object of inappropriate type.

eg:- 'Sum of 2 and 3' + 5
TypeError : ~~xxx~~ must be 'str' not 'int'

③ ValueError – This exception occurs whenever an inappropriate argument value even though it is of correct data type is used in a function call, then it will raise a value error.

eg- int('Hello')
ValueError : invalid literal for integer type casting.

④ ZeroDivisionError – This exception occurs when you try to perform numeric division in which denominator happens to be zero.

eg- >>> 78/(2+3-5)
ZeroDivisionError

⑤ OSError – This exception occurs whenever there is a system related error such as disk full or I/O error.

eg- f = open('temp.txt', 'r')
FileNotFound.

⑥ IndexError –

```
>>> ls = [1, 2, 3, 4]
>>> ls[10] = 100
```

raised when we try to access any index which is out of the valid range of iterator.

* Exception Handling -

→ To prevent a program from terminating abnormally, we need to handle it by catching the exception and taking appropriate action using 'try-except'.

→ try except :-

- A try block comprises of statements that have the potential to raise an exception.
- except block describes the action to be taken when an exception is raised.
- You can have multiple except statements attached to a single try block.
- You can write a single except statement to handle more than one kind of exception provided as an argument to the except statements.
- An empty except clause catches all possible exceptions.

Pg no - 234 (Write that example)

→ raise :-

- You can use raise keyword for raising an exception.

→ finally :-

⊙ You can use finally keyword to execute a set of instructions irrespective of whether an exception is raised or not.

Examples :-

1) Enter price of item purchased : 20
Enter weight of item purchased : x
Invalid inputs : ValueError

2) Enter price of item purchased : -20
Enter weight of item purchased : 10
(<class 'AssertionError'>, AssertionError(),
<traceback object at 0x0356D328>)

3) Enter price of item purchased : 20
Enter weight of item purchased : 0
Invalid inputs : (ZeroDivisionError)

4) Enter price of item purchased : 20
Enter weight of item purchased :
Invalid inputs : TypeError

5) Enter price of item purchased : -20
Enter weight of item purchased : 0
(<class 'AssertionError'>, AssertionError(),
<Traceback object at 0x03B42508>)

6) >>> print('Hello)
SyntaxError : EOL while scanning string literal

7) >>> for i in range(0,10)
SyntaxError : invalid syntax

8) >>> marks = Input ('Enter your marks')
Traceback (most recent call last):
File "<pyshell#0>", line 1, in <module>
marks = Input ('Enter your marks')
NameError : name 'Input' is not defined

Date - 06.01.24

1) def main():
try:
f = open ('Temporary-File', 'r')
except IOError:
print ('Problem with Input Output...')
print ('Program continues smoothly beyond try... except block')

if __name__ == '__main__':
main()

2) def main():
marks = 110
try: if marks < 0 or marks > 100:
raise ValueError ('Marks out of range')

```
finally:
    print('Bye')
print('Program continues after handling exception')

if __name__ == '__main__':
    main()
```

3)
```
def main():
    marks = 110
    try: if marks < 0 or marks > 100:
             raise ValueError('Marks out of range')
    except:
        pass
    finally:
        print('Bye')

if __name__ == '__main__':
    main()
```

1) Bubble Sort :-

```
def bubble_sort(lst):
    for i in range(0, len(lst)):
        for j in range(0, len(lst)-i):
            if (lst[j] > lst[j+1]):
                temp = lst[j]
                lst[j] = lst[j+1]
                lst[j+1] = temp
    return lst.
```

2) Insertion Sort :-

```
def insertion_sort (lst):
    n = len(lst)
    if n<=1:
        return
    for i in range(1, n):
        key = lst[i]
        j = i-1
    while j>=0 and key < lst[j]
        lst[j+1] = lst[j]
        j -= 1
    lst[j+1] = key
return lst
```

3) Selection Sort :-

```
def selection_sort (lst):
    for i in range (len(lst)):
        k = i
    for j in range (i+1, len(lst)):
        if lst[j] < lst[k]:
            k = j
    (lst[i], lst[k]) = (lst[k], lst[i])
return lst
```

4) Linear Search :-

```
def linear_search(lst, element):
    for i in range(0, len(lst)):
        if lst[i] == element:
            return i.
    return -1
```

5) Binary Search :-

```
def binary_search(lst, element):
    mid = len(lst)//2.
    if (lst[mid] == element):
        return mid
    elif (lst[mid] > element):
        return binary_search(lst[0:mid], element)
    else:
        return binary_search(lst[mid+1:], element)
```

Lab test - ch-2, 3, 6, 7, 8.