MODULE-2

PPT-4

Basic String Operations

- Method Description
- capitalize() Converts the first character to upper case
- casefold() Converts string into lower case
- center() Returns a centered string
- count() Returns the number of times a specified value occurs in a string
- encode() Returns an encoded version of the string
- endswith() Returns true if the string ends with the specified value
- expandtabs() Sets the tab size of the string
- find() Searches the string for a specified value and returns the position of where it was found

- format() Formats specified values in a string
- format_map() Formats specified values in a string
- index() Searches the string for a specified value and returns the position of where it was found
- isalnum() Returns True if all characters in the string are alphanumeric
- isalpha() Returns True if all characters in the string are in the alphabet
- isdecimal() Returns True if all characters in the string are decimals
- isdigit() Returns True if all characters in the string are digits
- isidentifier() Returns True if the string is an identifier
- islower() Returns True if all characters in the string are lower case
- isnumeric() Returns True if all characters in the string are numeric
- isprintable() Returns True if all characters in the string are printable
- isspace() Returns True if all characters in the string are whitespaces
- istitle() Returns True if the string follows the rules of a title
- isupper() Returns True if all characters in the string are upper case

- join() Joins the elements of an iterable to the end of the string
- ljust() Returns a left justified version of the string
- lower() Converts a string into lower case
- lstrip() Returns a left trim version of the string
- maketrans() Returns a translation table to be used in translations
- partition() Returns a tuple where the string is parted into three parts
- replace() Returns a string where a specified value is replaced with a specified value
- rfind() Searches the string for a specified value and returns the last position of where it was found
- rindex() Searches the string for a specified value and returns the last position of where it was found
- rjust() Returns a right justified version of the string
- rpartition() Returns a tuple where the string is parted into three parts
- rsplit() Splits the string at the specified separator, and returns a list

- rstrip() Returns a right trim version of the string
- split() Splits the string at the specified separator, and returns a list
- splitlines() Splits the string at line breaks and returns a list
- startswith() Returns true if the string starts with the specified value
- strip() Returns a trimmed version of the string
- swapcase() Swaps cases, lower case becomes upper case and vice versa
- title() Converts the first character of each word to upper case
- translate() Returns a translated string
- upper() Converts a string into upper case
- zfill() Fills the string with a specified number of 0 values at the beginning

Functions

- A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.
- In Python a function is defined using the def keyword:
- def my_function():
- print("Hello from a function")

To call a function, use the function name followed by parenthesis:

- def my_function():
- print("Hello from a function")
- my_function()

Arguments

- Information can be passed into functions as arguments.
- Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

```
def my_function(fname):
  print(fname + " Refsnes")
my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

- A parameter is the variable listed inside the parentheses in the function definition.
- An argument is the value that is sent to the function when it is called.

Arbitrary Arguments, *args

- If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.
- This way the function will receive a tuple of arguments, and can access the items accordingly:

```
def my_function(*kids):
  print("The youngest child is " + kids[2])
my_function("Emil", "Tobias", "Linus")
```

- You can also send arguments with the key = value syntax.
- This way the order of the arguments does not matter.
- def my_function(child3, child2, child1):
- print("The youngest child is " + child3)
- my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")

Default Parameter Value

- def my_function(country = "Norway"):
- print("I am from " + country)
- my_function("Sweden")
- my_function("India")
- my_function()
- my_function("Brazil")

Passing a List as an Argument

- You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

```
def my_function(food):
  for x in food:
    print(x)
fruits = ["apple", "banana", "cherry"]
my_function(fruits)
```

Return Values

- def my_function(x):
- return 5 * x
- print(my_function(3))
- print(my_function(5))
- print(my_function(9))

Write a Python program to reverse a string.

```
def string_reverse(str1):
    rstr1 = ''
    index = len(str1)
    while index > 0:
        rstr1 += str1[ index - 1 ]
        index = index - 1
    return rstr1
print(string_reverse('1234abcd'))
```

Write a Python function that takes a list and returns a new list with unique elements of the first list.

```
def unique_list(l):
  x = []
  for a in l:
    if a not in x:
      x.append(a)
  return x
print(unique_list([1,2,3,3,3,3,4,5]))
```

Practice question

- Write a Python function that checks whether a passed string is palindrome or not
- Write a Python function to calculate the factorial of a number (a non-negative integer). The function accepts the number as an argument.
- Write a Python function that accepts a string and calculate the number of upper case letters and lower case letters.

Recursion

- Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.
- The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.

Factorial

```
def factorial(x):
    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))
num = 3
print("The factorial of", num, "is", factorial(num))
```

Write a Python program to calculate the sum of a list of numbers.

```
def list_sum(num_List):
    if len(num_List) == 1:
        return num_List[0]
    else:
        return num_List[0] + list_sum(num_List[1:])

print(list_sum([2, 4, 5, 6, 7]))
```

Practice question using recursion

- Write a Python program to calculate the sum of the positive integers till a given number.
- Write a Python program to solve the Fibonacci sequence using recursion
- Write a Python program to calculate the harmonic sum of n elements.