

Author : DSALGO

PYTHON UNIT 3



string

- ◊ A string is a sequence of characters in Python.
- ◊ Formed by enclosing characters in quotes.
- ◊ Python treats single and double quotes equally.
- ◊ Strings are literal or scalar, treated as a single value in Python
- ◊ Python treats strings as a single data type.
- ◊ Allows easy access and manipulation of string parts.

Example :

```
str1 = 'Hello World'  
str2 = "My name is khan"
```

String manipulation in Python

1.Concatenation:

Joining two or more strings together.

```
result = str1 + " " + str2  
print(result) # Output: "Hello World My name is khan"
```

2.String Formatting:

Creating formatted strings with placeholders or f-strings.

```
fstr = f"{str1} and {str2}"  
print(fstr)
```

3.String Methods:

Using built-in string methods to manipulate strings.

```
u_str = str1.upper() #HELLO WORLD  
l_str = str1.lower() # hello world  
r_str = str2.replace("khan ", "singh")#my name is singh
```

4.Splitting and Joining:

Splitting a string into a list of substrings based on a delimiter, and joining a list of strings into a single string.

```
splitlist = str1.split("") #['Hello','World']  
joinlist = "-".join(splitlist) # Hello – World
```

5.Stripping:

Removing leading and trailing whitespace characters from a string.

```
my_string = " Hello World "  
stripped_str = my_string.strip()  
print(stripped_str) # Output: "Hello World"
```

6.Searching:

Finding substrings within a string.

```
string = "the quick brown fox jumps over the lazy dog"  
is_dog = "dog" in string #bool:true or false  
index = string.find("fox") #return first occurrence index
```

slicing string manipulation :

- ◊ string slicing is a technique used to extract a portion of a string by specifying a range of indices.
- ◊ **Syntax :**
`string[start:stop:step]`
- ◊ **Example:**

Basic slicing

```
substring1 = text[7:10] # Extracts "is"  
substring2 = text[0:6] # Extracts "Python"
```

Omitting start or stop

```
substring3 = text[:6] # Extracts "Python"  
substring4 = text[7:] # Extracts "is amazing"
```

Using negative indices

```
substring5 = text[-7:-1] # Extracts "amazin"
```

Slicing with step

```
substring6 = text[0:15:2] # Extracts "Pto saa"
```

Reverse a string

```
substring7 = text[::-1] # Extracts "gnizama si nohtyP"
```

list

- ◊ list is a mutable, ordered collection of element.
- ◊ Lists are one of the most versatile & widely used data types in Python.

Here are some key characteristics of lists:

Creation :

```
my_list = [1, 2, 3, "hello", True]
```

Indexing:

```
first_element = my_list[0] # Accessing the first elementssing the first element
```

Slicing :

```
sublist = my_list[1:4] # Extracting elements from index 1 to 3
```

Mutability:

```
my_list[0] = 100 # Modifying an element  
my_list.append(5) # Adding an element to the end  
my_list.remove("hello") # Removing an element
```

Length:

```
length_of_list = len(my_list)
```

Methods:

```
my_list.sort() # Sorts the list in ascending order  
my_list.reverse() # Reverses the list
```

copy() method in list :

```
original_list = [1, 2, 3, 4, 5]  
copied_list1 = original_list.copy() #method 1  
print(copied_list1)  
copied_list2 = list(original_list) # method 2  
print(copied_list2)  
copied_list3 = original_list[:] # method 3  
print(copied_list3)
```

Delete element in list

```
my_list = [1, 2, 3, 4, 5]  
del my_list[2] # Removes the element at index 2  
print(my_list) #[1, 2, 4, 5]  
my_list = [1, 2, 3, 4, 5]  
my_list.remove(3) # Removes the first occurrence of the value 3  
print(my_list) #[1, 2, 4, 5]  
my_list = [1, 2, 3, 4, 5]  
popped_element = my_list.pop(2) # Removes and returns the element at index 2  
print(my_list) #[1, 2, 4, 5]  
my_list = [1, 2, 3, 4, 5]  
my_list[1:3] = [] # Deletes elements at indices 1 and 2  
print(my_list) #[1, 4, 5]  
my_list = [1, 2, 3, 4, 5]  
my_list.clear() # Removes all elements from the list  
print(my_list) # []
```

Built-in operator in python :

Concatenation

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
result_list = list1 + list2 # Concatenates the lists
```

Repetition

```
repeated_string = "abc" * 3 # Repeats the string three times
repeated_list = [1, 2] * 4 # Repeats the list four times
In operator
string_check = "lo" in "Hello" # Checks if "lo" is in "Hello" (True)
list_check = 3 in [1, 2, 3, 4] # Checks if 3 is in the list (True)
```

tuple

- ❖ tuple is an ordered, immutable collection of elements.
- ❖ once a tuple is created, its elements cannot be modified or changed.
- ❖ Tuples are defined using parentheses () and can contain elements of different data types.
- ❖ **Program :**

```
my_tuple = (1, 2, 3, 'hello', True) # tuple creation
first_element = my_tuple[0] #Accessing Elements
my_tuple[0] = 100 # This will raise an error
packed_tuple = 1, 'apple', 3.14 #Tuple Packing
x, y, z = packed_tuple #Unpacking
length = len(my_tuple) Length of Tuple
combined_tuple = (1, 2, 3) + ('a', 'b', 'c') #Tuple Concatenation
count_of_2 = my_tuple.count(2) #output : 1
index_of_hello = my_tuple.index('hello') #output: 3
```

Dictionary

- ❖ dictionary is a mutable, unordered collection of key-value pairs.
- ❖ Each key in a dictionary must be unique, and it is associated with a specific value.
- ❖ Dictionaries are defined using curly braces {} and consist of key-value pairs separated by commas.
- ❖ Here are some key points about dictionaries:
 - ❖ Creating a Dictionary:
 - ❖ my_dict = {'name': 'John', 'age': 25, 'city': 'New York'}
- ❖ **Accessing Values:**

```
name = my_dict['name']
age = my_dict['age']
```
- ❖ **Modifying Values:**

```
my_dict['age'] = 26
```
- ❖ **Adding New Key-Value Pairs:**

```
my_dict['gender'] = 'Male'
```
- ❖ **Removing Key-Value Pairs:**

```
del my_dict['city']
```
- ❖ **Checking Key Existence:**

```
is_name_present = 'name' in my_dict
```
- ❖ **Dictionary Methods:**

```
keys = my_dict.keys()
values = my_dict.values()
items = my_dict.items()
```

Set

- ❖ set is an unordered and mutable collection of unique elements.
- ❖ Sets are defined using curly braces {} and can contain various data types, such as numbers, strings, and other sets.
- ❖ Operations performed on sets are union , intersection , difference , symmetric difference

Program :

Defining sets

```
set1 = set([1, 2, 4, 1, 2, 8, 5, 4])
set2 = set([1, 9, 3, 2, 5])
```

Printing sets

```
print(set1) # Output: {1, 2, 4, 5, 8}
```

```
print(set2) # Output: {1, 2, 3, 5, 9}
```

Intersection of set1 and set2

```
intersection = set1 & set2
```

```
print(intersection) # Output: {1, 2, 5}
```

Union of set1 and set2

```
union = set1 | set2
```

```
print(union) # Output: {1, 2, 3, 4, 5, 8, 9}
```

Difference of set1 and set2

```
difference = set1 - set2
```

```
print(difference) # Output: {4, 8}
```

Symmetric difference of set1 and set2

```
symm_diff = set1 ^ set2
```

```
print(symm_diff) # Output: {3, 4, 8, 9}
```

Functions

- ❖ Functions are self-contained units designed for specific tasks.
- ❖ Once created, functions can be called as needed.
- ❖ Functions have names and may or may not return values.
- ❖ Python offers built-in functions like dir(), len(), abs(), etc.
- ❖ Users can define custom functions for specific requirements.
- ❖ We will use the def keyword to define functions

Why do we use functions :(advantages)

- | | |
|-------------------|-----------------|
| 1.Reusability | 2. Readability, |
| 3.Maintainability | 4. Scoping |

Syntax:

```
# define the function_name
def function_name():
```

```
    # Code inside the function
    # Optionally, return a value
```

#call the function_name

```
Function_name()
```

Example function without parameters

```
def greet():
    print("Hello, World!")
greet() # Calling the greet function
```

Example function with parameters

```
def add_numbers(a, b):
    return a + b
```

Calling the add_numbers function

```
result = add_numbers(5, 7)
print("Sum:", result)
```

Elements of the function

- ◊ **Keyword:** def for function header.
- ◊ **Function Name:** Identifies uniquely.
- ◊ **Colon:** Ends function header.
- ◊ **Parameters:** Values passed at a defining time , e.g., x and y.
- ◊ **Arguments:** Values passed at a calling time , e.g., a and b.
- ◊ **Function Body:** Processes arguments.
- ◊ **Return Statement:** Optionally returns a value.
- ◊ **Function Call:** Executes the function, e.g., a = max(8, 6).
- ◊ **Keyword Arguments:**Pass values to function by associating them with parameter names.
Syntax : function_name(p1=val1 , p2 =val2)
- ◊ **Default Values:** can have default values, making them optional during function calls.
Syntax: def function_name(param1=default_value1, param2=default_value2);

scope rules in Python [LEGB rule]

- Local:** Variables within a function; accessible only within that function.
- Enclosing:** Scope for nested functions; refers to the enclosing function's scope.
- Global:** Variables at the top level of a module or declared global within a function; accessible throughout the module.
- Builtin:** Outermost scope containing built-in names like print(), sum(), etc.; available everywhere.

```
# Global scope
global_variable = "I am global"
def outer_function():
    # Enclosing scope
    enclosing_variable = "I am in the enclosing scope"
    def inner_function():
        # Local scope
        local_variable = "I am in the local scope"
        print(local_variable)
        print(enclosing_variable)
        print(global_variable)
        print(len(global_variable)) # Using a built-in function
    inner_function()
outer_function()
```

Output :

```
I am in the local scope
I am in the enclosing scope
I am global
10
```

Q . make a calculator using function inpython like sub, add, mult, div

```
# four basic Functions for calculator sub, sum , divide, multiply
def add(x, y):
    return x + y
def sub(x, y):
    return x - y
def mult(x, y):
    return x * y
def div(x, y):
    if y != 0:
        return x / y
    else:
        return "Error! Division by zero."
```

```
#Make a table for selection operation
print("Select operation:")
print("1. Add")
print("2. Subtract")
print("3. Multiply")
print("4. Divide")
choice = input("Enter choice (1/2/3/4): ")
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
if choice == '1':
    print(num1, "+", num2, "=", add(num1, num2))
elif choice == '2':
    print(num1, "-", num2, "=", sub(num1, num2))
elif choice == '3':
    print(num1, "*", num2, "=", mult(num1, num2))
elif choice == '4':
    print(num1, "/", num2, "=", div(num1, num2))
else:
    print("Invalid input. Please enter a valid choice.")
```

Q. Write a function in find the HCF of given numbers

```
def HCF(num1, num2):
    while num2:
        num1, num2 = num2, num1 % num2
    return abs(num1)
# Example usage:
number1 = 24
number2 = 36
result = HCF(number1, number2)
print(f"The HCF of {number1} and {number2} is {result}")
```