**Module 13 Assignment 7 Functions and methods**

**Defining and calling functions in Python.**

**Python function :**

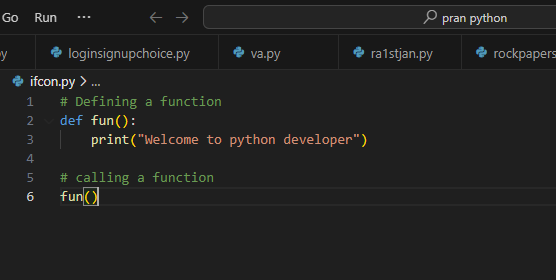
**Python Functions** is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.

**Some Benefits of Using Functions**

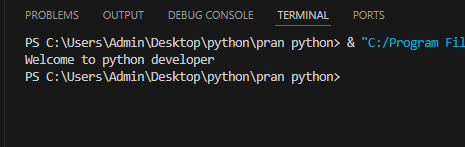
* Increase Code Readability
* Increase Code Reusability

In Python, defining and calling functions is simple and may greatly improve the readability and reusability of our code.

**Example:**



**Output:**



**Defining a Function**

By using the word def keyword followed by the function's name and parentheses () we can define a function. If the[function](https://www.geeksforgeeks.org/python-functions/) takes any[arguments](https://www.geeksforgeeks.org/default-arguments-in-python/), they are included inside the parentheses. The code inside a function must be indented after the colon to indicate it belongs to that function.

### ****Syntax of defining a function:****

def function\_name(parameters):

# Code to be executed

return value

* **function\_name:**The name of your function.
* **parameters:** Optional. A list of parameters (input values) for the function.
* **return:**Optional. The return statement is used to send a result back from the function.

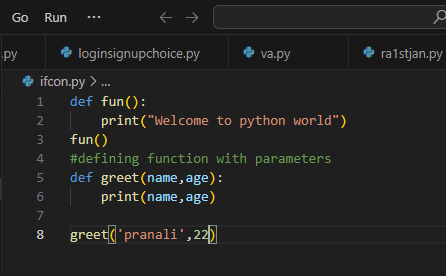
## ****Calling a Function****

To call a function in Python, we definitely type the name of the function observed via parentheses (). If the function takes any arguments, they may be covered within the parentheses . Below is the example for calling def function Python.

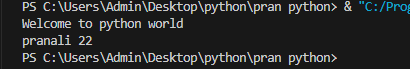
### ****Syntax of Calling a function:****

function\_name(arguments)

Example:



Output:



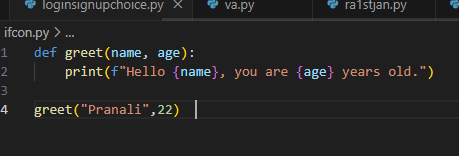
Function arguments (positional, keyword, default)

In Python, functions can accept arguments in three main ways: positional arguments, keyword arguments, and default arguments. Here's a breakdown of each:

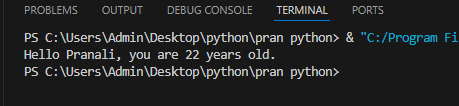
### 1. ****Positional Arguments****

These are the most basic type of arguments, passed in the order they appear in the function definition. The number of arguments passed must match the number of parameters in the function definition.

Example:



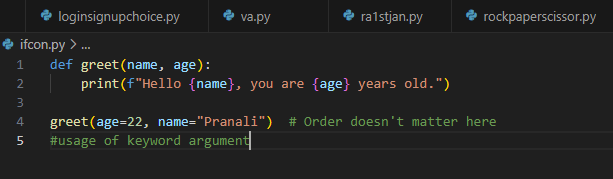
Output:

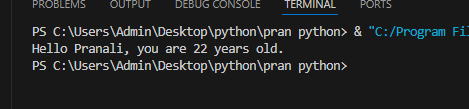


### 2. ****Keyword Arguments****

In this case, arguments are passed by explicitly specifying the name of the parameter along with its value. This allows you to pass arguments in any order, and the function knows which argument corresponds to which parameter.

Example:

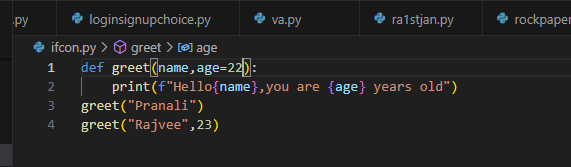




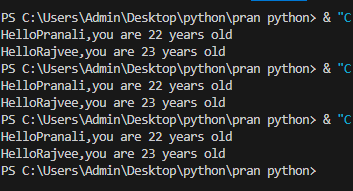
### 3. ****Default Arguments****

Default arguments are used when you want to provide a default value for a parameter, in case the caller does not provide a value for that parameter. If the caller does not pass a value for the parameter, the default value is used.

Example:



Output:

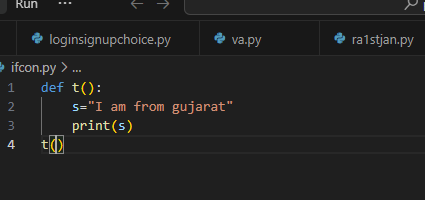


Python scope variable

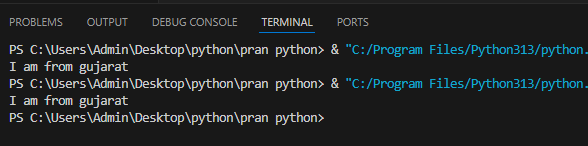
The location where we can find a variable and also access it if required is called the **scope of a variable**.

### Python Local variable

Local variables are those that are initialized within a function and are unique to that function. It cannot be accessed outside of the function. Let’s look at how to make a local variable.



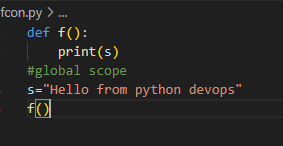
Output:



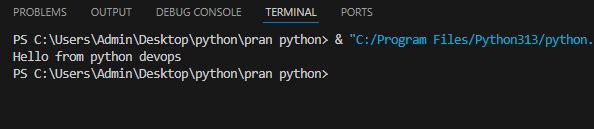
### Python Global variables

Global variables are the ones that are defined and declared outside any function and are not specified to any function. They can be used by any part of the program.

Example:



Output:



| **Method** | **Description** | **Example** | **Output** |
| --- | --- | --- | --- |
| capitalize() | Capitalizes the first character, converts others to lowercase. | "hello".capitalize() | "Hello" |
| casefold() | Converts the string to lowercase, used for case-insensitive comparison. | "HELLO".casefold() | "hello" |
| center(width, fillchar) | Centers the string within a specified width, fills with fillchar. | "hello".center(10, "\*") | "\*\*hello\*\*\*" |
| count(substring) | Returns the number of non-overlapping occurrences of a substring. | "hello world".count("l") | 3 |
| encode(encoding) | Encodes the string to bytes using the specified encoding. | "hello".encode("utf-8") | b'hello' |
| endswith(suffix) | Checks if the string ends with the specified suffix. | "hello world".endswith("world") | True |
| expandtabs(tabsize) | Replaces tabs with spaces, using a specified tab size. | "hello\tworld".expandtabs(4) | "hello world" |
| find(substring) | Returns the lowest index of the substring, or -1 if not found. | "hello world".find("world") | 6 |
| format(\*args) | Formats the string by inserting arguments in placeholders. | "Hello {}".format("Alice") | "Hello Alice" |
| index(substring) | Like find(), but raises ValueError if substring is not found. | "hello world".index("world") | 6 |
| isalpha() | Returns True if all characters in the string are alphabetic. | "hello".isalpha() | True |
| isascii() | Returns True if all characters are ASCII. | "hello".isascii() | True |
| isdigit() | Returns True if all characters are digits. | "12345".isdigit() | True |
| islower() | Returns True if all characters are lowercase. | "hello".islower() | True |
| isupper() | Returns True if all characters are uppercase. | "HELLO".isupper() | True |
| join(iterable) | Joins elements of an iterable with the string as a separator. | "-".join(["a", "b", "c"]) | "a-b-c" |
| lstrip(chars) | Removes leading whitespace or specified characters. | " hello".lstrip() | "hello" |
| lower() | Converts all characters to lowercase. | "HELLO".lower() | "hello" |
| replace(old, new) | Replaces occurrences of a substring with another substring. | "hello world".replace("world", "Python") | "hello Python" |
| split(sep) | Splits the string at the separator into a list of substrings. | "apple,banana,grape".split(",") | ['apple', 'banana', 'grape'] |
| strip(chars) | Removes leading and trailing whitespace or specified characters. | " hello ".strip() | "hello" |
| swapcase() | Converts lowercase characters to uppercase and vice versa. | "Hello World".swapcase() | "hELLO wORLD" |
| title() | Converts the first character of each word to uppercase. | "hello world".title() | "Hello World" |
| upper() | Converts all characters to uppercase. | "hello".upper() | "HELLO" |
| zfill(width) | Pads the string with zeros on the left to reach the specified width. | "42".zfill(5) | "00042" |
| partition(separator) | Splits the string into a 3-tuple at the first occurrence of separator. | "hello world".partition(" ") | ('hello', ' ', 'world') |

.

Built-in List Methods

| **Method** | **Description** | **Example** | **Output** |
| --- | --- | --- | --- |
| append(x) | Adds an element x to the end of the list. | lst = [1, 2, 3]; lst.append(4) | [1, 2, 3, 4] |
| extend(iterable) | Extends the list by appending all elements from the iterable. | lst = [1, 2]; lst.extend([3, 4]) | [1, 2, 3, 4] |
| insert(i, x) | Inserts element x at index i. | lst = [1, 2, 3]; lst.insert(1, 10) | [1, 10, 2, 3] |
| remove(x) | Removes the first occurrence of element x. | lst = [1, 2, 3, 2]; lst.remove(2) | [1, 3, 2] |
| pop([i]) | Removes and returns the element at index i (last item if i is not provided). | lst = [1, 2, 3]; lst.pop() | 3 |
| clear() | Removes all elements from the list. | lst = [1, 2, 3]; lst.clear() | [] |
| index(x) | Returns the index of the first occurrence of element x. | lst = [1, 2, 3]; lst.index(2) | 1 |
| count(x) | Returns the number of occurrences of element x. | lst = [1, 2, 2, 3]; lst.count(2) | 2 |
| sort() | Sorts the list in ascending order (modifies the list in place). | lst = [3, 1, 2]; lst.sort() | [1, 2, 3] |
| reverse() | Reverses the elements of the list in place. | lst = [1, 2, 3]; lst.reverse() | [3, 2, 1] |
| copy() | Returns a shallow copy of the list. | lst = [1, 2, 3]; new\_lst = lst.copy() | [1, 2, 3] |
| join(iterable) | Joins elements of an iterable into a string (works with strings). | lst = ['a', 'b', 'c']; '-'.join(lst) | 'a-b-c' |
| sort(reverse=True) | Sorts the list in descending order. | lst = [3, 1, 2]; lst.sort(reverse=True) | [3, 2, 1] |
| sorted() | Returns a new list with elements sorted (does not modify the original). | lst = [3, 1, 2]; sorted(lst) | [1, 2, 3] |
| clear() | Removes all elements from the list. | lst = [1, 2, 3]; lst.clear() | [] |
| copy() | Returns a shallow copy of the list. | lst = [1, 2, 3]; lst\_copy = lst.copy() | [1, 2, 3] |
| min() | Returns the smallest element in the list. | lst = [3, 1, 2]; min(lst) | 1 |
| max() | Returns the largest element in the list. | lst = [3, 1, 2]; max(lst) | 3 |
| sum() | Returns the sum of all elements in the list. | lst = [1, 2, 3]; sum(lst) | 6 |