TOPS Technology

Python Fundamentals

Presented By:

Nandni Vala

Functions and Methods

1.Defining and calling functions in Python.

- functions are reusable blocks of code that perform a specific task. They are defined using the def keyword and can be called by using their name followed by parentheses.
- **Defining a Function**
- Here's the basic syntax for defining a function:
- def function_name(parameters):
- """
- Optional docstring describing the function.
- """
- # Function body
- return Value# Optional

- **function_name**: The name of the function.
- **parameters**: Values passed to the function (optional).
- **return**: The output of the function (optional).
- **Calling a Function :**
- To execute a function, write its name followed by parentheses and provide arguments if required.
- **Syntax:**
- function_name(arguments)

2.Function arguments (positional, keyword, default).

Positional Arguments

These are the most common type of arguments. Their values are assigned to parameters in the order they are defined in the function signature.

Example:

- def greet(name, age):
- print(f"My name is {name} and I am {age} years old.")
- # Call the function using positional arguments
- greet("Vala", 21)
- **Output:**
- My name is Vala and I am 25 years old.
- **Keyword Arguments**
- In this case, arguments are passed using parameter names, regardless of their order in the function definition.
- **Example:**
- def greet(name, age):
- print(f"My name is {name} and I am {age} years old.")
- # Call the function using keyword arguments
- greet(age=21, name="Vala")

- Output:
- My name is Vala and I am 25 years old.
- **Default Arguments**
- Default arguments allow you to specify default values for parameters. If the caller does not provide a value for a parameter, the default value is used.
- **Example:**

```
def greet(name="Guest", age=18):
    print(f"My name is {name} and I am {age} years old.")
# Call the function with both arguments
greet("Vala", 21)
# Call the function with one argument
greet("Nandni")
# Call the function with no arguments
greet()
```

- Output:
- My name is Vala and I am 21 years old.
- My name is Nandni and I am 18 years old.
- My name is Guest and I am 18 years old.

3.Scope of variables in Python.

- The **scope of a variable** determines where that variable can be accessed or modified.
- Python has a well-defined hierarchy of variable scopes that follows the **LEGB Rule** (Local, Enclosing, Global, Built-in).
- **LEGB Rule**: Hierarchy of Scopes
- **Local (L)**: Variables defined inside a function or block. They are accessible only within that function or block.
- **Enclosing (E)**: Variables in the scope of the enclosing (outer) function, useful in nested functions.
- **Global (G)**: Variables defined at the top level of a script or module, accessible throughout the module unless shadowed by local variables.
- **Built-in (B)**: Variables and functions provided by Python, such as len(), print(), or range().

4.Built-in methods for strings, lists, etc.

- Python provides a rich set of built-in methods for strings, lists, dictionaries, sets, and other data types.

- **String Methods :** String methods allow you to manipulate and analyze text.

Method	Description	Example
str.lower()	Converts string to lowercase	`"Hello".lower()` → `'hello'`
str.upper()	Converts string to uppercase	`"hello".upper()` → `'HELLO'`
str.capitalize()	Capitalizes the first character	`"python".capitalize()` → `'Python'`
str.strip()	Removes leading and trailing whitespace	`" hello ".strip()` → `'hello'`
str.replace(old, new)	Replaces occurrences of a substring	`"hello".replace("l", "z")` → `'hezzo'`
str.split(separator)	Splits string into a list of substrings	`"a,b,c".split(",")` → `['a', 'b', 'c']`
str.join(iterable)	Joins elements of an iterable with the string	`",".join(['a', 'b', 'c'])` → `'a,b,c'`
str.find(sub)	Returns the index of the first occurrence of `sub`	`"hello".find("l")` → `2`
str.isdigit()	Checks if all characters are digits	`"123".isdigit()` → `True`

➢**List Methods :** List methods help manipulate list elements.

Method	Description	Example
list.append(item)	Adds an item to the end of the list	lst.append(4)
list.extend(iterable)	Adds all items of an iterable to the list	lst.extend([4, 5])
list.insert(index, item)	Inserts an item at a specific index	lst.insert(1, "a")
list.remove(item)	Removes the first occurrence of an item	lst.remove(3)
list.pop(index)	Removes and returns the item at `index` (last by default)	`lst.pop(1)` → Removes second item
list.index(item)	Returns the index of the first occurrence of `item`	lst.index(2)
list.sort()	Sorts the list in ascending order (in-place)	lst.sort()
list.reverse()	Reverses the list in-place	lst.reverse()
list.count(item)	Counts occurrences of `item` in the list	lst.count(2)

➢**Dictionary Methods :**

➢Dictionary methods allow you to work with key-value pairs.

Method	Description	Example
dict.keys()	Returns a view object of keys	`d.keys()` → `dict_keys(['a', 'b'])`
dict.values()	Returns a view object of values	`d.values()` → `dict_values([1, 2])`
dict.items()	Returns a view object of key-value pairs	`d.items()` → `dict_items([('a', 1), ...])`
dict.get(key, default)	Returns the value for a key, or `default` if missing	d.get('a', 0)
dict.update(other_dict)	Updates dictionary with key-value pairs from another	d.update({'c': 3})
dict.pop(key)	Removes and returns the value for `key`	d.pop('a')
dict.clear()	Removes all items from the dictionary	d.clear()

- **Set Methods :**
- Set methods handle operations on unique elements.

Method	Description	Example
set.add(item)	Adds an item to the set	s.add(3)
set.remove(item)	Removes an item; raises `KeyError` if not found	s.remove(3)
set.discard(item)	Removes an item; does nothing if not found	s.discard(3)
set.union(other)	Returns the union of sets	s.union({3, 4})
set.intersection(other)	Returns the intersection of sets	s.intersection({2, 3})
set.difference(other)	Returns the difference of sets	s.difference({3})
set.clear()	Removes all items from the set	s.clear()

Tuple Methods :

Tuples are immutable, so they have fewer methods.

Method	Description	Example
tuple.count(item)	Counts occurrences of `item`	t.count(2)
tuple.index(item)	Returns the index of the first occurrence of `item`	t.index(3)