MASTER OF COMPUTER APPLICATIONS (MCA)

Post Graduate Department of Computer Science and Technology

Sardar Patel University

Vallabh Vidyanagar, Gujarat, INDIA

# PS01CMCA51 [Python Programming]

## Unit-III

MR. BHARAT B. PATEL (ASSOCIATE PROFESSOR)

(✉) bb_patel@spuvvn.edu

# Mapping type - dict

- Python is a completely object-oriented language.
- You have been working with classes and objects right from the beginning of these tutorials. Every element in a Python program is an object of a class.
- A number, string, list, dictionary, etc., used in a program is an object of a corresponding built-in class.
- You can retrieve the class name of variables or objects using the type() method, as shown below.

•

# Mapping type - dict

- A dictionary is a collection which is ordered [When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change] / unordered [means that the items does not have a defined order, you cannot refer to an item by using an index], changeable and do not allow duplicates.
- All the list objects are the objects of the dict class in Python.
- Use the dict() constructor is use to declare dictionary object.
- A dictionary items can be iterated using the for loop.

# Mapping type - dict

## Creating a Dictionary

- In Python, a Dictionary can be created by placing a sequence of elements within curly {} braces, separated by 'comma'.
- Dictionary holds pairs of values, one being the Key and the other corresponding pair element being its Key:value.
- Values in a dictionary can be of any data type and can be duplicated, whereas keys can't be repeated and must be immutable.
- Note – Dictionary keys are case sensitive, the same name but different cases of Key will be treated distinctly.
- Dictionary can also be created by the built-in function dict().
- An empty dictionary can be created by just placing to curly braces{}.

# Example of dict

```
>>> dct1=dict ( )
>>> dct2= {}
>>> type ( dct1)
<class 'dict'>
>>> type ( dct2)
<class 'dict'>
>>> dct1= { 1:"One", 2:"Two", 3:"Three", 4:"Four"}
>>> print ( dct1)
 { 1: 'One', 2: 'Two', 3: 'Three', 4: 'Four'}
>>> print ( dct1[2])
Two
>>> print ( dct1["2"])
Traceback  ( most recent call last)  :
  File "<pyshell#7>", line 1, in <module>
    print ( dct1["2"])
KeyError: '2'
>>>
```

# Python Dictionary Methods

| Dictionary Method | Description |
|---|---|
| dict.clear() | Removes all the key-value pairs from the dictionary. |
| dict.copy() | Returns a shallow copy of the dictionary. |
| dict.fromkeys() | Creates a new dictionary from the given iterable (string, list, set, tuple) as keys and with the specified value. |
| dict.get() | Returns the value of the specified key. |
| dict.items() | Returns a dictionary view object that provides a dynamic view of dictionary elements as a list of key-value pairs. This view object changes when the dictionary changes. |
| dict.keys() | Returns a dictionary view object that contains the list of keys of the dictionary. |
| dict.pop() | Removes the key and return its value. If a key does not exist in the dictionary, then returns the default value if specified, else throws a KeyError. |
| dict.popitem() | Removes and return a tuple of (key, value) pair from the dictionary. Pairs are returned in Last In First Out (LIFO) order. |

# Python Dictionary Methods

| Dictionary Method | Description |
|---|---|
| dict.setdefault() | Returns the value of the specified key in the dictionary. If the key not found, then it adds the key with the specified defaultvalue. If the defaultvalue is not specified then it set None value. |
| dict.update() | Updates the dictionary with the key-value pairs from another dictionary or another iterable such as tuple having key-value pairs. |
| dict.values() | Returns the dictionary view object that provides a dynamic view of all the values in the dictionary. This view object changes when the dictionary changes. |
| dict.clear() | Removes all the key-value pairs from the dictionary. |
| dict.copy() | Returns a shallow copy of the dictionary. |
| dict.fromkeys() | Creates a new dictionary from the given iterable (string, list, set, tuple) as keys and with the specified value. |
| dict.get() | Returns the value of the specified key. |
| dict.items() | Returns a dictionary view object that provides a dynamic view of dictionary elements as a list of key-value pairs. This view object changes when the dictionary changes. |

# Python Dictionary Methods

| Dictionary Method | Description |
|---|---|
| dict.keys() | Returns a dictionary view object that contains the list of keys of the dictionary. |
| dict.pop() | Removes the key and return its value. If a key does not exist in the dictionary, then returns the default value if specified, else throws a KeyError. |
| dict.popitem() | Removes and return a tuple of (key, value) pair from the dictionary. Pairs are returned in Last In First Out (LIFO) order. |
| dict.setdefault() | Returns the value of the specified key in the dictionary. If the key not found, then it adds the key with the specified defaultvalue. If the defaultvalue is not specified then it set None value. |
| dict.update() | Updates the dictionary with the key-value pairs from another dictionary or another iterable such as tuple having key-value pairs. |
| dict.values() | Returns the dictionary view object that provides a dynamic view of all the values in the dictionary. This view object changes when the dictionary changes. |

# Python set type

- A set is a mutable collection of distinct hashable objects, same as the list and tuple.
- It is an unordered collection of objects, meaning it does not record element position or order of insertion and so cannot access elements using indexes.
- The set is a Python implementation of the set in Mathematics.
- A set object has suitable methods to perform mathematical set operations like union, intersection, difference, etc.
- A set object contains one or more items, not necessarily of the same type, which are separated by a comma and enclosed in curly brackets {}.
- A set doesn't store duplicate objects. Even if an object is added more than once inside the curly brackets, only one copy is held in the set object.
- Hence, indexing and slicing operations cannot be done on a set object.

# Python set type

- The order of elements in the set is not necessarily the same as the order given at the time of assignment.
- Python optimizes the structure of a set for performing operations over it, as defined in mathematics.
- Only immutable (and hashable) objects can be a part of a set object.
- Numbers (integer, float, as well as complex), strings, and tuple objects are accepted, but set, list, and dictionary objects are not.
- The set() function is used to convert string, tuple, or dictionary object to a set object.

# Mapping type - dict

- **Creating set objects :**

```
>>> set1 = set ()
>>> set2 = set ( {2,2,2,12,12,12} )
>>> type ( set1)
<class 'set'>
>>> type ( set2)
<class 'set'>
>>> set3 = { ( 1,2) , ( 3,4) , "Test", True, 12, "a",}
>>> len ( set3)
6
```

# Python Set Methods

| Dictionary Method | Description |
|---|---|
| set.add() | Adds an element to the set. If an element is already exist in the set, then it does not add that element. |
| set.clear() | Removes all the elements from the set. |
| set.copy() | Returns a shallow copy of the set. |
| set.difference() | Returns the new set with the unique elements that are not in the another set passed as a parameter. |
| set.discard() | Removes a specific element from the set. |
| set.intersection() | Returns a new set with the elements that are common in the given sets. |
| set.isdisjoint() | Returns true if the given sets have no common elements. Sets are disjoint if and only if their intersection is the empty set. |

# Python Set Methods

| Dictionary Method | Description |
|---|---|
| set.issubset() | Returns true if the set (on which the issubset() is called) contains every element of the other set passed as an argument. |
| set.pop() | Removes and returns a random element from the set. |
| set.remove() | Removes the specified element from the set. If the specified element not found, raise an error. |
| set.union() | Returns a new set with distinct elements from all the given sets. |
| set.update() | Updates the set by adding distinct elements from the passed one or more iterables. |

# Python set type

- The order of elements in the set is not necessarily the same as the order given at the time of assignment.
- Python optimizes the structure of a set for performing operations over it, as defined in mathematics.
- Only immutable (and hashable) objects can be a part of a set object.
- Numbers (integer, float, as well as complex), strings, and tuple objects are accepted, but set, list, and dictionary objects are not.
- The set() function is used to convert string, tuple, or dictionary object to a set object.

# Functions in Python

- **Python includes many built-in functions.**
- **These functions perform a predefined task and can be called upon in any program, as per requirement.**
- **However, if you don't find a suitable built-in function to serve your purpose, you can define it.**

**What is Function?**

- **A function is a reusable block of programming statements designed to perform a certain task.**

**OR**

- **Python Functions is a block of related statements designed to perform a specific task (*computational, logical, or evaluative task*).**

# Functions in Python

| Built-in functions | User-defined functions (UDFs) | Recursive functions | Lamda (Anonymous) functions |
|---|---|---|---|
| • Already defined or available in library. | • Defined by the user. | • Calling function to itself. | • Function defined in a single line without actually naming it. |

# Python Functions

- To define a function, Python provides the def keyword.
- Functions can be both built-in or user-defined.
- It helps the program to be concise, non-repetitive, and organized.

**Syntax**

```
def  function_name(parameters):
        """ docstring """
        statement-1
        statement-2
        :
        :
        return [expr]
```

Pass statement

# Python Functions

- **def** - **marks the start of the function header.**
- **function_name** - **to uniquely identify the function. (Function naming follows the same rules of writing identifiers in Python.)**
- **parameters** - **used to pass values to a function. (optional)**
  ***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.***
- **colon (:)** - **to mark the end of the function header.**
- **docstring** - **(document string) to describe what the function does. (optional)**
- **statements** - **Makes/define the body of the function. (same indentation)**
- **return** - **to return a value from the function. (optional)**

# Functions in Python

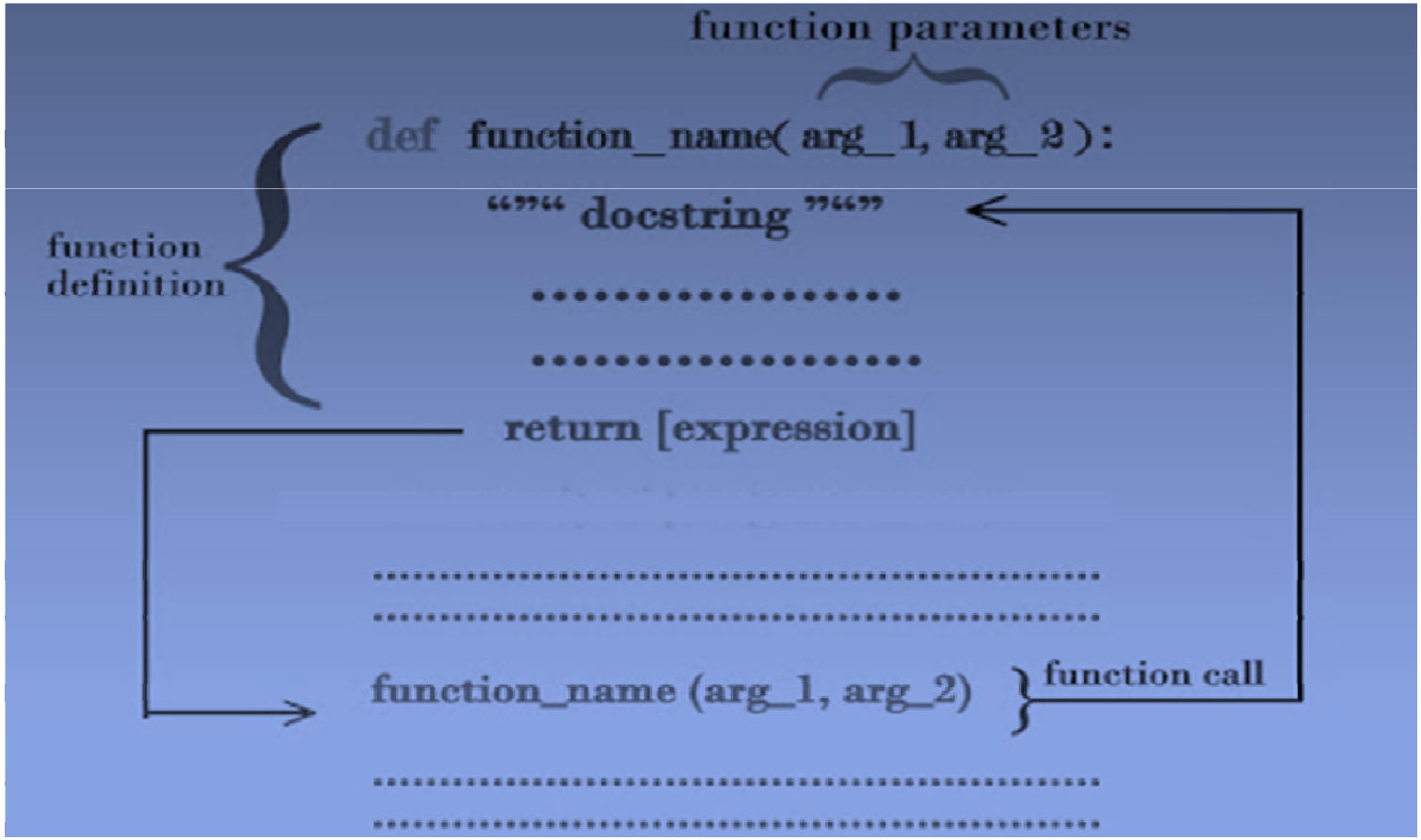
**How to call a function?**

- **Once we have defined a function, we can call it from another function, program, or even the Python prompt.**
- **To call a function we simply type the function name with appropriate parameters.**
- ***For example :*** **>>>add(5,4)**

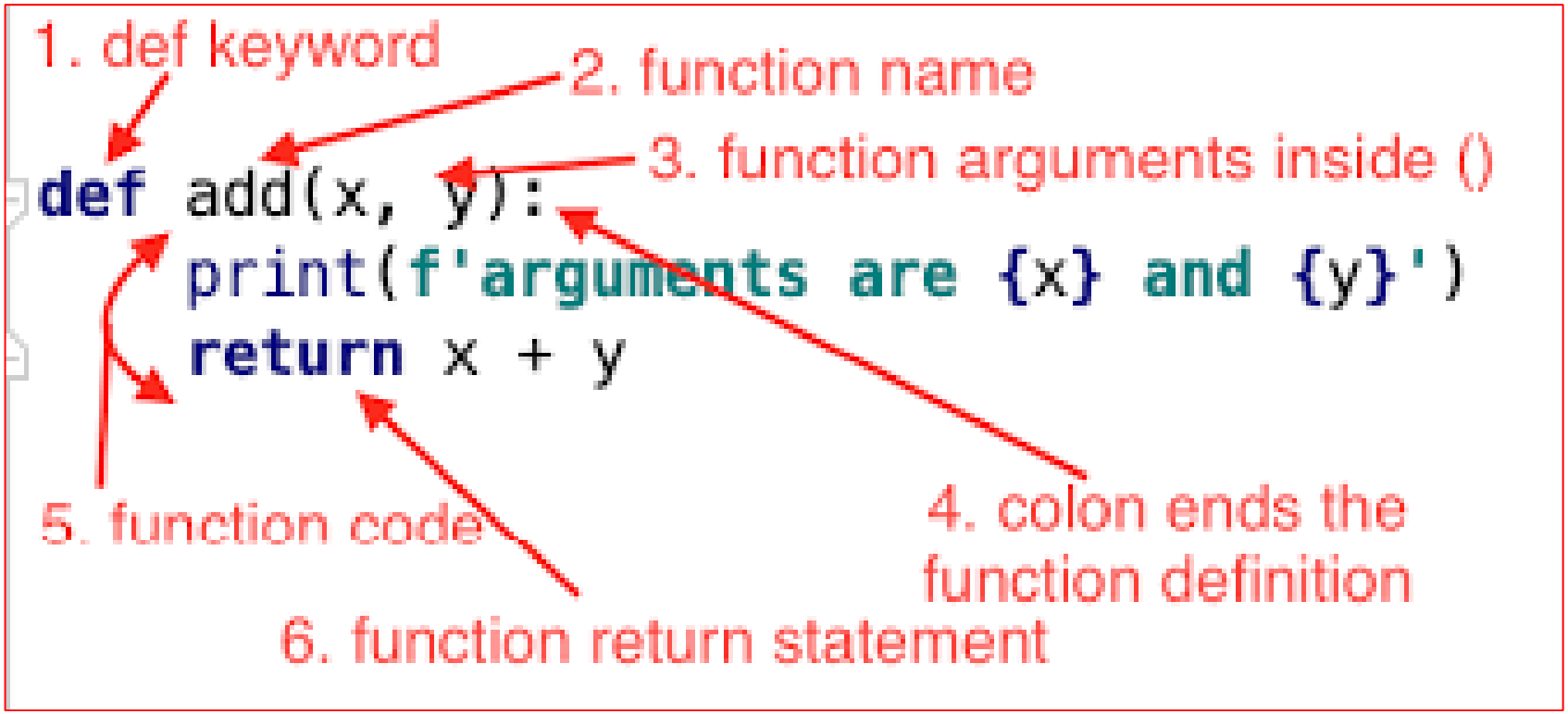
**Benefits of using functions :**

- **To break our large program into smaller and modular chunks.**
- **To manage and organize the large code.**
- **To avoid repetition.**
- **To makes the code reusable.**
- **To reduce the development time.**

# Python Functions



# Python Functions



# Exceptions in Python

- An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.
- In general, when a Python script encounters a situation that it cannot cope with, it raises an exception.
- An exception is a Python object that represents an error.
- When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

# Exceptions in Python

## Handling an Exception

- If you have some suspicious code that may raise an exception, you can defend your program by placing the suspicious code in a *try:* block. After the *try:* block, include an *except:* statement, followed by a block of code which handles the problem as elegantly as possible.

```
try:
    #statements in try block
except:
    #executed when error in try block
else:
    #executed if try block is error-free
finally:
    #executed irrespective of exception occured or not
```

# Exceptions in Python

- Python uses *try* and *except* keywords to handle exceptions.
- Both keywords are followed by indented blocks.
- The *try:* block contains one or more statements which are likely to encounter an exception.
- If the statements in this block are executed without an exception, the subsequent *except:* block is skipped.
- If the exception does occur, the program flow is transferred to the *except:* block.
- The statements in the *except:* block are meant to handle the cause of the exception appropriately. For example, returning an appropriate error message.

# Exceptions in Python

- You can specify the type of exception after the except keyword.
- The subsequent block will be executed only if the specified exception occurs.
- There may be multiple except clauses with different exception types in a single try block.
- If the type of exception doesn't match any of the except blocks, it will remain unhandled and the program will terminate.
- The rest of the statements after the except block will continue to be executed, regardless if the exception is encountered or not.

# Exceptions in Python

- The following example will throw an exception when we try to devide an integer by a string.

```
try:
    a=5
    b=0
    print(a/b)
except:
    print("Some exception found")
print("Out of try block")
```

- Output :

```
Some exception found
Out of try block
```

# Exceptions in Python

- A single try block may have multiple except blocks.
- The following example uses two except blocks to process two different exception types:

```
try:
    a=5
    b=0
    print(a/b)
except TypeError:
    print("Some Typing related Exception")
except ZeroDivisionError:
    print("Can'd Divide by Zeor")

print("Out of try block")
```

- Output

```
Can'd Divide by Zeor
Out of try block
```

# Exceptions in Python

- In Python, keywords else and finally can also be used along with the try and except clauses.
- While the except block is executed if the exception occurs inside the try block, the else block gets processed if the try block is found to be exception free.
- The finally block consists of statements which should be processed regardless of an exception occurring in the try block or not.
- As a consequence, the error-free try block skips the except clause and enters the finally block before going on to execute the rest of the code.
- If, however, there's an exception in the try block, the appropriate except block will be processed, and the statements in the finally block will be processed before proceeding to the rest of the code.

# Built-in or Standard Exceptions in Python

| Sr. No. | Exception Name & Description | |
|---|---|---|
| 1 | Exception | Base class for all exceptions |
| 2 | StopIteration | Raised when the next() method of an iterator does not point to any object. |
| 3 | SystemExit | Raised by the sys.exit() function. |
| 4 | StandardError | Base class for all built-in exceptions except StopIteration and SystemExit. |
| 5 | ArithmeticError | Base class for all errors that occur for numeric calculation. |
| 6 | OverflowError | Raised when a calculation exceeds maximum limit for a numeric type. |
| 7 | FloatingPointError | Raised when a floating point calculation fails. |
| 8 | ZeroDivisionError | Raised when division or modulo by zero takes place for all numeric types. |
| 9 | AssertionError | Raised in case of failure of the Assert statement. |
| 10 | AttributeError | Raised in case of failure of attribute reference or assignment. |
| 11 | EOFError | Raised when there is no input from either the raw_input() or input() function and the end of file is reached. |
| 12 | ImportError | Raised when an import statement fails. |
| 13 | KeyboardInterrupt | Raised when the user interrupts program execution, usually by pressing Ctrl+c. |
| 14 | LookupError | Base class for all lookup errors. |
| 15 | IndexError | Raised when an index is not found in a sequence. |
| 16 | KeyError | Raised when the specified key is not found in the dictionary. |
| 17 | NameError | Raised when an identifier is not found in the local or global namespace. |
| 18 | UnboundLocalError | Raised when trying to access a local variable in a function or method but no value has been assigned to it. |
| 19 | EnvironmentError | Base class for all exceptions that occur outside the Python environment. |
| 20 | IOError | Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist. |

# Built-in or Standard Exceptions in Python

| Sr. No. | Exception Name & Description | |
|---|---|---|
| 21 | IOError | Raised for operating system-related errors. |
| 22 | SyntaxError | Raised when there is an error in Python syntax. |
| 23 | IndentationError | Raised when indentation is not specified properly. |
| 24 | SystemError | Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit. |
| 25 | SystemExit | Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit. |
| 26 | TypeError | Raised when an operation or function is attempted that is invalid for the specified data type. |
| 27 | ValueError | Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified. |
| 28 | RuntimeError | Raised when a generated error does not fall into any category. |
| 29 | NotImplementedError | Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented. |

# Exceptions in Python

***How to raise an exception?***

- **Python also provides the *raise* keyword to be used in the context of exception handling.**
- **It causes an exception to be generated explicitly.**
- **Built-in errors are raised implicitly.**
- **However, a built-in or custom exception can be forced during execution.**
- **As a Python developer you can choose to throw an exception if a condition occurs.**
- **To throw (or raise) an exception, use the *raise* keyword.**