Programming Refresher

Python Functions

Learning Objectives

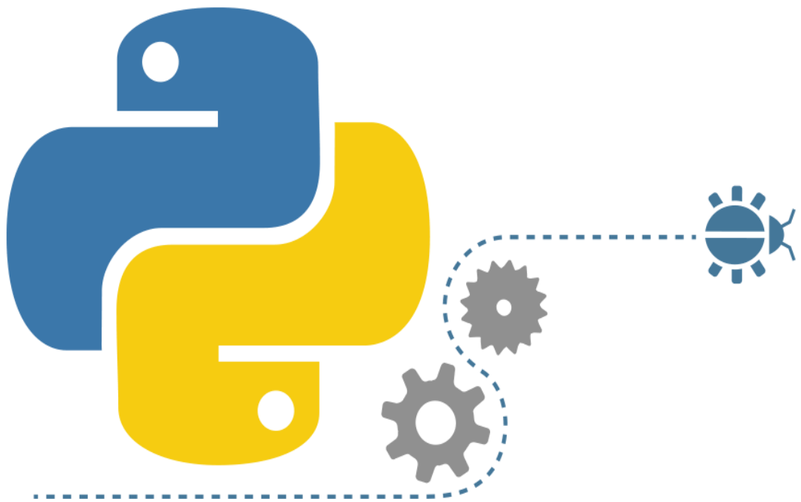
By the end of this lesson, you will be able to:

- Illustrate the concept of functions
- Summarize the advantages of functions
- Design a function with arguments
- Explicate the scope of a variable in a function
- Summarize generators

DATA AND
ARTIFICIAL INTELLIGENCE

Functions and Its Advantages

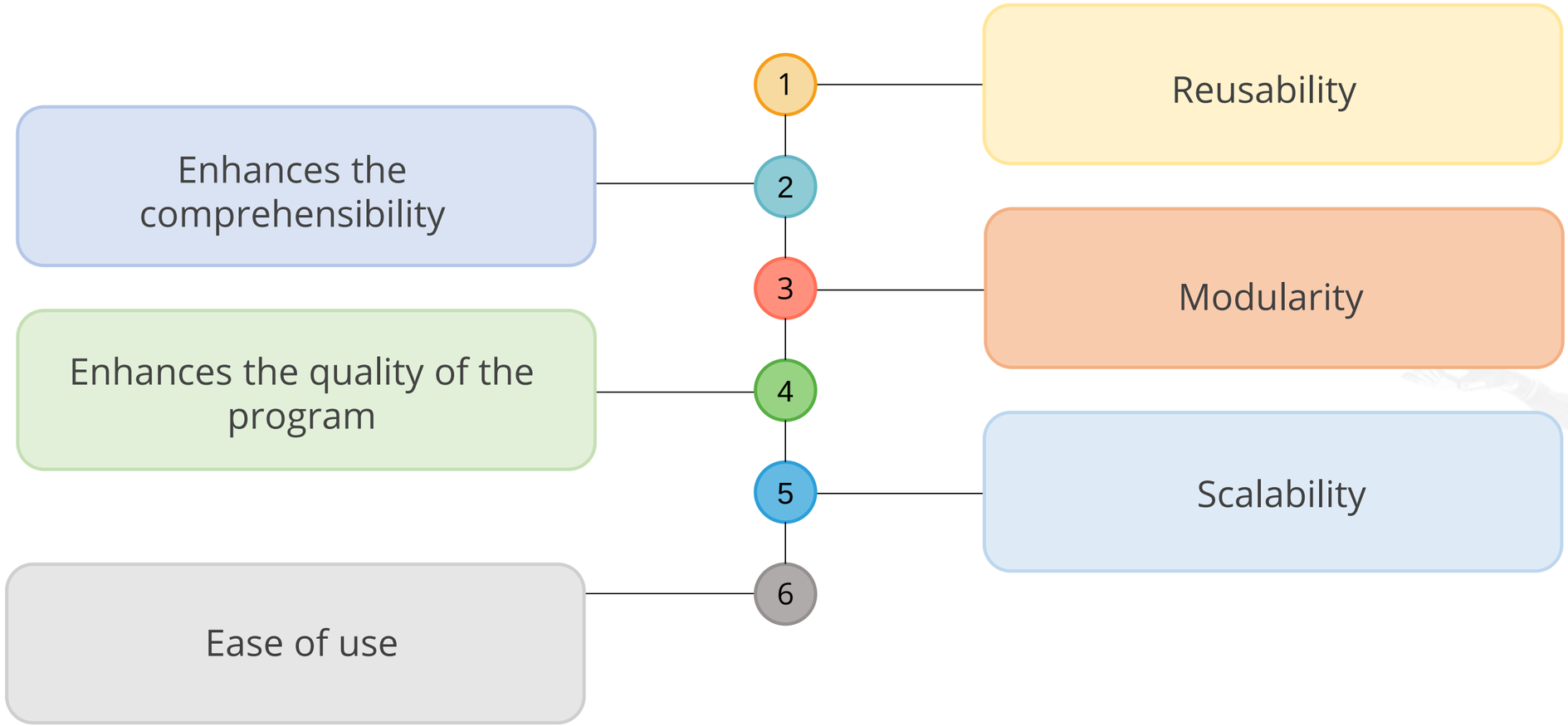
Python: Functions



- A function is a collection of connected statements that achieves a certain goal.
- It may be defined as the organized block of reusable code.
- It is a code block that only executes when it is invoked.
- Parameters are data that are passed into a function.

Functions: Advantages

The advantages of functions are:



Functions: Syntax

The basic syntax of a function in Python consists of two parts:

01

Function definition

- A function is defined using the keyword *def* followed by a function name and parenthesis.
- The argument names passed to the function are mentioned inside this parenthesis.
- The body of the function is an indented block of code.

Functions: Syntax

The basic syntax of a function in Python consists of two parts:

02

Function call

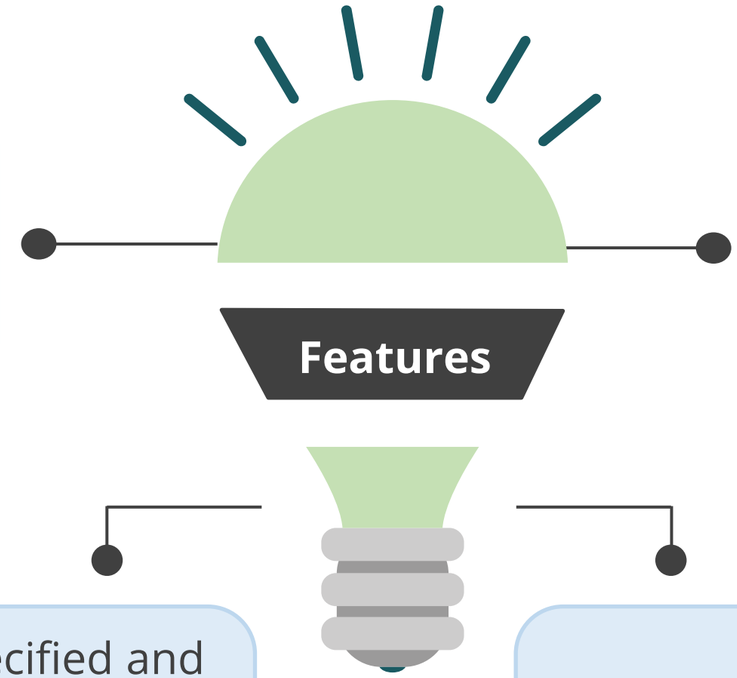
- A function can be called from anywhere in the program in order to be used.
- When a function is called, the program control is transferred to the called function to perform the defined task.
- All the necessary parameters and arguments should be passed during the function call.

DATA AND
ARTIFICIAL INTELLIGENCE

Function Arguments

Functions: Arguments

The features of the arguments are mentioned below:



Information can be passed to a function in the form of arguments.

A function may have zero to multiple arguments separated by a comma.

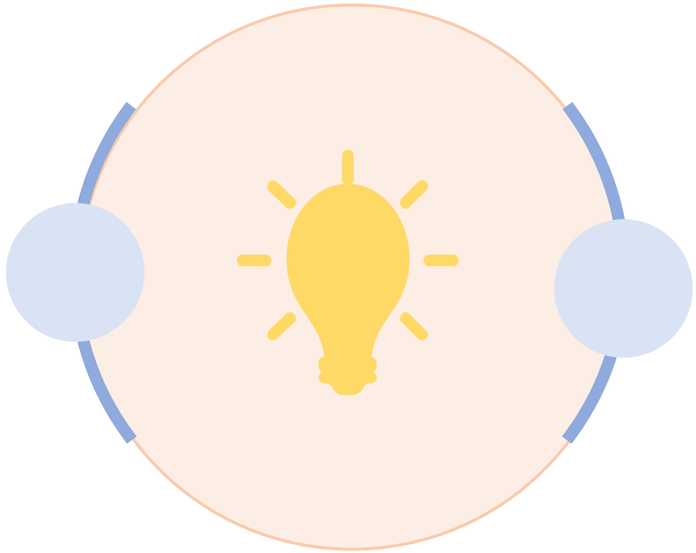
The arguments are specified and enclosed in parentheses in the function definition immediately after the function name.

The arguments can be data values, objects, or functions.

Functions: Argument Matching

Arguments in Python can be matched by position or name.

Positional arguments are required to be mentioned in the same order as in the function definition.



Keyword arguments are referenced by name and may be in any order.

Positional and keyword arguments can be mixed in a function call.

Note

Arguments in a function may also be defined with a default value.

Assisted Practice: Function and Argument Matching

Duration: 5 mins

Objective: In this demonstration, you will learn to create a function with arguments.

Steps to perform:

Step 1: Create a function

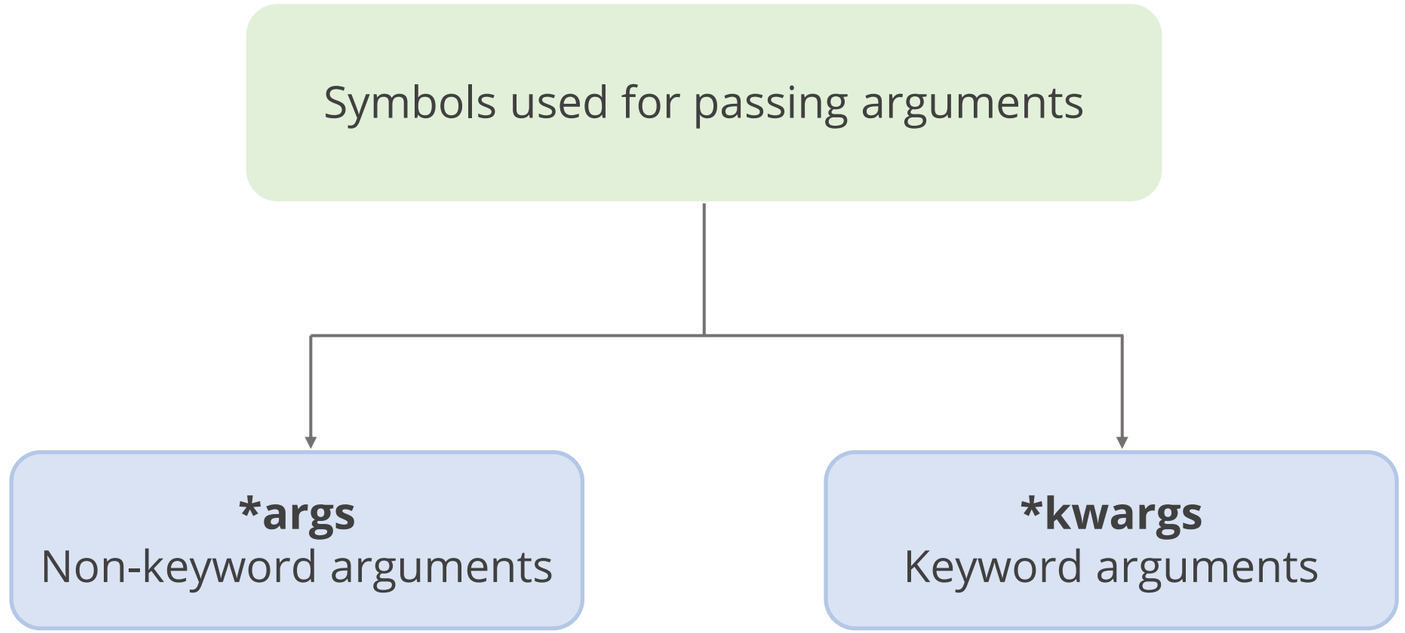
Step 2: Pass arguments to the function

Step 3: Perform argument matching

Step 4: Call the function and display the results

Functions: Arbitrary Arguments

Python allows the passing of variables with multiple arguments by using special symbols.



Note

These notations are used when the number of arguments to be passed to the function is uncertain.

Functions: Arbitrary Arguments

The types of arbitrary arguments are explained below:

Non-keyword arguments	Keyword arguments
The symbol * is used to take in a variable number of arguments.	Keyword arguments are defined using **. Conventionally, “kwargs” is used to define the keyword arguments.
Conventionally, “args” is used to define the non-keyword argument.	A keyword argument allows giving the variable a name before passing it to the function.
These arguments are unpacked as a list inside the function.	Keyword arguments are like a dictionary where each value is mapped to the keyword passed along with it.

Assisted Practice: Arbitrary Arguments

Duration: 5 mins

Objective: In this demonstration, you will learn to use arbitrary arguments.

Steps to perform:

Step 1: Define a function that takes arbitrary arguments

Step 2: Pass arguments to the function

Step 3: Call the function and display the results

return Statement

Functions: return Statement

The *return* keyword is used to pass values back to the function call.

If the *return* statement is not used, Python returns *None* by default.

After the *return* statement is executed, no further statements of the function are executed.

A function can return none to multiple values or data objects.

Assisted Practice: return Statement

Duration: 5 mins

Objective: In this demonstration, you will learn to use the *return* statement.

Steps to perform:

Step 1: Define a function with arguments

Step 2: Use the *return* statement to return values when the function is called

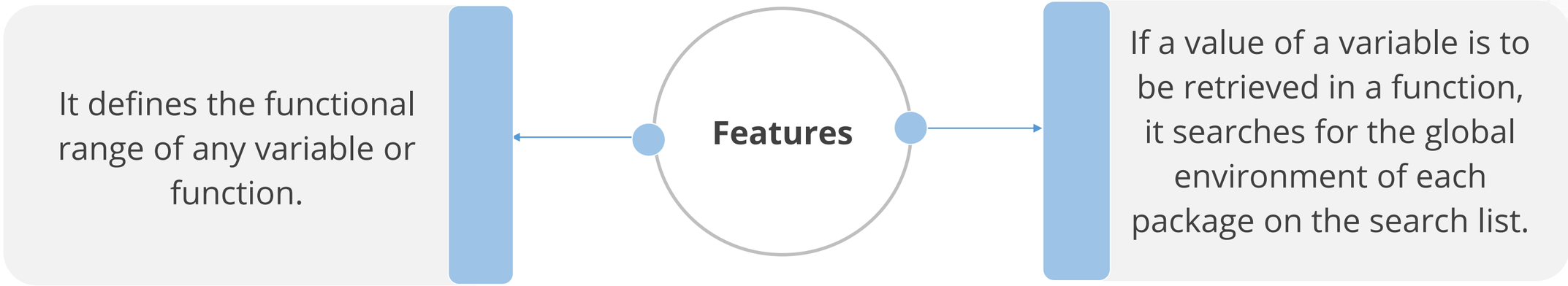
Step 3: Call the function and display the results

DATA AND
ARTIFICIAL INTELLIGENCE

Scope of a Variable

Function: Scope

Scope in programming defines the environment where the variables can be accessed and referenced.



Function: Scope

There are two types of scope:

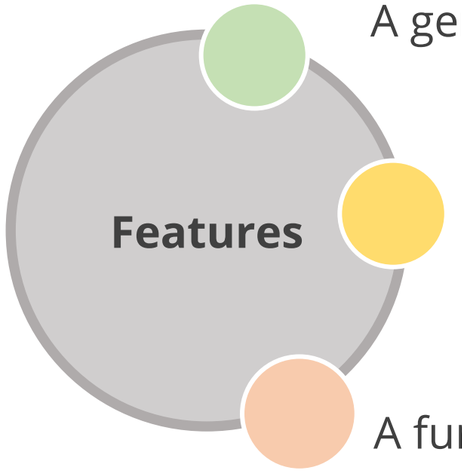
Global scope	Local scope
Variables and functions defined outside of all classes and functions have global scope.	Variables or functions that are defined inside a function block including argument names, are local variables.
It allows the variable values to be used or functions to be called from anywhere in the file in the program.	It can be accessed only inside the function block.

DATA AND
ARTIFICIAL INTELLIGENCE

Generators Function

Function: Generators

Generators are a special type of function supported in Python that returns an iterator object with a sequence of values.



A generator uses a *yield* statement instead of a return statement.

A generator object can be iterated only once.

A function with a *yield* statement is termed a generator.

return vs. yield Statement

The difference between the *return* and *yield* statements are given below:

return statement

- The *return* statement implies that the function is returning control of execution to the point from where the function is called.
- The process destroys the local variable and values generated.

yield statement

- The *yield* statement implies that the transfer of control is temporary.
- Hence, it does not destroy the states of the local variables of the function.

Generator: Example

Here a generator function is defined that yields three values:

```
def myGenerator():
    print('First iteration')
    yield 'Number 1'

    print('Second iteration')
    yield 'Number 2'

    print('Third iteration')
    yield 'Number 3'
```

Note

In place of multiple *yield* statements, a *loop* can also be used inside the function to generate the values.

Ways to Use a Generator

The generator function can be used in two ways :

It can be used with the built-in *next* function.

```
gen = myGenerator()
print(next(gen))
print('\nNext :')
print(next(gen))
print('\nNext :')
print(next(gen))
# repeat until all values are yielded
```

```
First iteration
Number 1

Next :
Second iteration
Number 2

Next :
Third iteration
Number 3
```

It can be used with a *for* loop as an iteration object.

```
gen = myGenerator() # generator object
for i in gen:
    print(i)
    print('Next iteration\n')
```

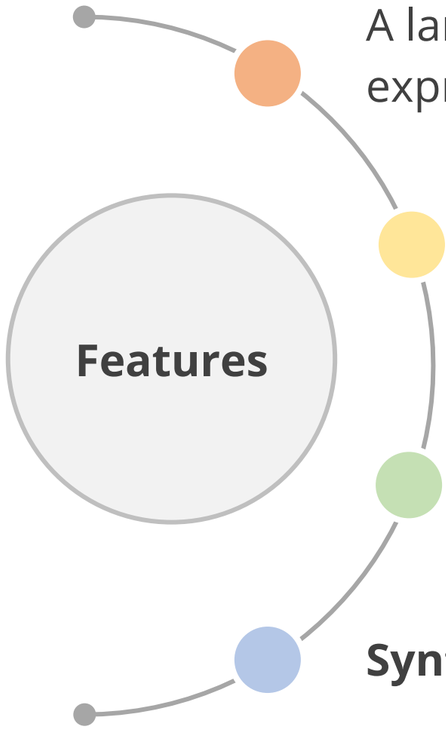
```
First iteration
Number 1
Next iteration

Second iteration
Number 2
Next iteration

Third iteration
Number 3
Next iteration
```

DATA AND
ARTIFICIAL INTELLIGENCE

Lambda Functions

Lambda Functions

Python lambda functions are anonymous functions and can be defined by using the keyword *lambda.*



A lambda function may have any number of arguments but contains only one expression.

The lambda functions are syntactically restricted to only a single expression which is evaluated and returned.

It is possible to define a lambda function without assigning it to a variable or explicitly calling it.

Syntax: lambda arguments: expression

DATA AND
ARTIFICIAL INTELLIGENCE

Function Types

Types of Functions

Functions can be of two types:

Built-in functions	User-defined functions
Built–in functions are predefined functions in the programming framework.	Python also supports the creation of customized user-defined functions to perform specific tasks.
Python has basic built-in functions, such as len(), sum(), type(), slice(), next(), help(), format().	A user can give any function name with any number of arguments.

DATA AND
ARTIFICIAL INTELLIGENCE

Functional Programming Implementation

Functional Programming Methods

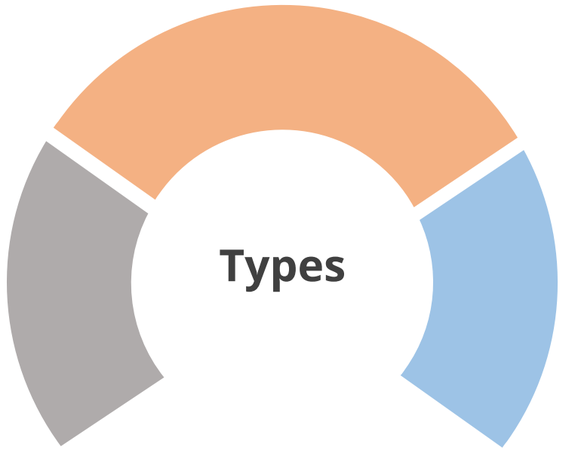
These built-in functions facilitate functional programming in Python, which uses methods to define computation.

Filter function

Syntax: filter(func, iterable)

Map function

Syntax: map(func, iterable)



Reduce function

Syntax: reduce(func, iterable)

The *map* and *filter* functions are basic built-in functions, whereas the *reduce* function is a part of a module named "_functools."

map()

A *map* function can be used to apply a function to each element of an iteration object.

Example

```
def fahrenhite(T):
    return (( 9/5 * T )+32)

temperatures = [22, 45, 25, 30]
res = list(map(fahrenhite, temperatures))
res

[71.6, 113.0, 77.0, 86.0]
```

```
temperatures = [22, 45, 25, 30]
res = list(map(lambda T: ((9/5 * T)+32), temperatures))
res

[71.6, 113.0, 77.0, 86.0]
```

- The *map* function takes in two arguments a function and an iterable object.
- It applies the given function to all elements of the given iteration object.
- It generates an iterator which can be converted to a list.
- It can also use the *lambda* function for implementation.

filter()

A *filter* function is used to filter data based on a condition defined in a function.

Example

```
numbers = [37, 90, 81, 24, 75, 31, 53, 12]
odd_numbers = list(filter(lambda x: x % 2, numbers))
print(odd_numbers)
```

```
[37, 81, 75, 31, 53]
```

- The *filter* function also takes a function and an iterable object and applies the function to each element of the iteration object.
- The given function should return a Boolean value.

reduce()

A *reduce* function is used to implement the mathematical technique of folding on an iteration object.

Example

```
from _functools import reduce
numbers = [2,4,6,8,10]
res = reduce(lambda x,y : x + y, numbers)
print(res)
```

```
30
```

- The *reduce* function takes in a function an iterable object.
- The reduce function applies the function continually to each element of the iterable object and produces a single cumulative value.

Key Takeaways

- Functions are an important structure of any programming language to create a modular and reusable application.
- Python functions use the *def* keyword to define functions, can take 0 or more arguments, and can return a value or None.
- Variables declared in the function have local scope and variables declared outside a function have global scope.
- Iterators are a way to traverse over a string, list, or tuple and they exhibit the concept of generators.
- The Lambda keyword is used to define an anonymous function in Python.

DATA AND
ARTIFICIAL INTELLIGENCE

Thank You