# Python Programming Language Foundation

Session 4

## Session overview

- Functions
- Lambda functions
- Scopes

# Functions

## Function definition

A **function** is a **block** of *organized, reusable* code that is used to perform a single, related action.

*https://www.tutorialspoint.com/python/python_functions*

## Function definition

```
def function_name([arguments]):
    """optional_function_docstring"""
    [function_suite]
    return [expression]
```

## Function definition

```
def function_name([arguments]):
    """optional_function_docstring"""
    [function_suite]
    return [expression]
```

Indentation MATTERS!

## Function call

```
function_name([arguments])
```

## Function arguments

```
function_name([ANY_OBJECT])
```

## Argument types

Positional

Keyword

## Argument types

| Positional | Keyword |
|---|---|
| Positional arguments | Keyword arguments |
| Positional with default value | Keyword without default value |
| Tuple of positional arguments | Dictionary of keyword arguments |

## Positional arguments

```
def function_name(a, b, c, d):
```

## Positional arguments

```
function_name(1, 2, 3, 4)
```

## Positional arguments

```
function_name(a=1, b=2, c=3, d=4)
```

## Positional arguments

```
function_name(b=1, d=2, c=3, a=4)
```

## Positional arguments

```
def function_name(a, b, c, *d):
```

## Positional arguments

```
function_name(1, 2, 3, 4, 5, 6, 7)
```

## Positional arguments

```
function_name(b=1, c=3, a=4)
```

## Positional arguments

```
function_name(b=1, c=3, a=4, d=1)

# TypeError: function_name got
an unexpected keyword argument 'd'
```

## Positional arguments

```
function_name(b=1, c=3, a=4, 1, 2, 3)

# SyntaxError: positional
argument follows keyword argument
```

## Keyword arguments

```
def function_name(a, b, c=True, d=False):
```

## Keyword arguments

```
function_name(1, 2)
```

## Keyword arguments

```
function_name(1, 2, False, True)
```

## Keyword arguments

```
function_name(b=1, d=True, c=False, a=4)
```

## Keyword arguments

```
def function_name(a, b, c=True, **d):
```

## Keyword arguments

```
function_name(1, 1, c=False, k=100, n=another_function)
```

## Arguments confusion

```
def function_name(a, b=1, *c, d=2, e, **f): #?
```

*https://medium.com/@boxed/keyword-argument-confusion-in-python-59105c5a1159*

## Arguments confusion

```
def function_name(a, b=1, *c, d=2, e, **f):
```

```
# positional argument
```

*https://medium.com/@boxed/keyword-argument-confusion-in-python-59105c5a1159*

## Arguments confusion

```python
def function_name(a, b=1, *c, d=2, e, **f):
# positional argument with default value
```

https://medium.com/@boxed/keyword-argument-confusion-in-python-59105c5a1159

## Arguments confusion

```
def function_name(a, b=1, *c, d=2, e, **f):

                    # args
```

[https://medium.com/@boxed/keyword-argument-confusion-in-python-59105c5a1159](https://medium.com/@boxed/keyword-argument-confusion-in-python-59105c5a1159)

## Arguments confusion

```
def function_name(a, b=1, *c, d=2, e, **f):

                    # keyword argument
```

https://medium.com/@boxed/keyword-argument-confusion-in-python-59105c5a1159

## Arguments confusion

```
def function_name(a, b=1, *c, d=2, e, **f):

  # keyword argument without default value
```

https://medium.com/@boxed/keyword-argument-confusion-in-python-59105c5a1159

## Arguments confusion

```python
def function_name(a, b=1, *c, d=2, e, **f):

                    # kwargs
```

https://medium.com/@boxed/keyword-argument-confusion-in-python-59105c5a1159

## Arguments order

```python
def function_name(pos, default=1, *args, key=2, without, **kwargs):
```

## PEP 570

```python
def function_name(a, b, c=3, /):
```

https://www.python.org/dev/peps/pep-0570/

## PEP 570

```
function_name(a=1, b=2, c=3)

# TypeError
```

https://www.python.org/dev/peps/pep-0570/

## Arguments unpacking

```python
def handle_info(name, age, sex, friends):
```

## Arguments unpacking

```
person_info = ('Bob', 27)

person_additional_info = {'sex': 'male', 'friends': ('Kate',)}

handle_info(*person_info, **person_additional_info)

handle_info('Bob', 27, sex='male', friends=('Kate',))
```

## Built-in Functions

| Built-in Functions | | | | |
|---|---|---|---|---|
| abs() | delattr() | hash() | memoryview() | set() |
| all() | dict() | help() | min() | setattr() |
| any() | dir() | hex() | next() | slice() |
| ascii() | divmod() | id() | object() | sorted() |
| bin() | enumerate() | input() | oct() | staticmethod() |
| bool() | eval() | int() | open() | str() |
| breakpoint() | exec() | isinstance() | ord() | sum() |
| bytearray() | filter() | issubclass() | pow() | super() |
| bytes() | float() | iter() | print() | tuple() |
| callable() | format() | len() | property() | type() |
| chr() | frozenset() | list() | range() | vars() |
| classmethod() | getattr() | locals() | repr() | zip() |
| compile() | globals() | map() | reversed() | __import__() |
| complex() | hasattr() | max() | round() | |

https://docs.python.org/3/library/functions.html

## Lambda functions

## Anonymous Function

```
lambda [arguments]: expression
```

## Scopes

## Scopes

**Scope** is rule how **variables** and **names** are looked up in your code.

*https://realpython.com/python-scope-legb-rule/*

## Scopes

```python
total = 0  # global

def sum(arg1, arg2):
    """Sum the parameters and return the result."""
    total = arg1 + arg2  # local
    print(f"Inside the function: {total}")

sum(10, 20)
print(f"Outside the function: {total}")
```

## Scopes

```
total = 0  # global

def sum(arg1, arg2):
    """Sum the parameters and return the result."""
    total = arg1 + arg2  # local
    print(f"Inside the function: {total}")

sum(10, 20)  # 30
print(f"Outside the function: {total}")  # 0
```

## Scopes

```
total = 0  # global

def sum(arg1, arg2):
    """Sum the parameters and return the result."""
    result = (arg1 + arg2) * total  # local
    print(f"Inside the function: {result}")

sum(10, 20)
```

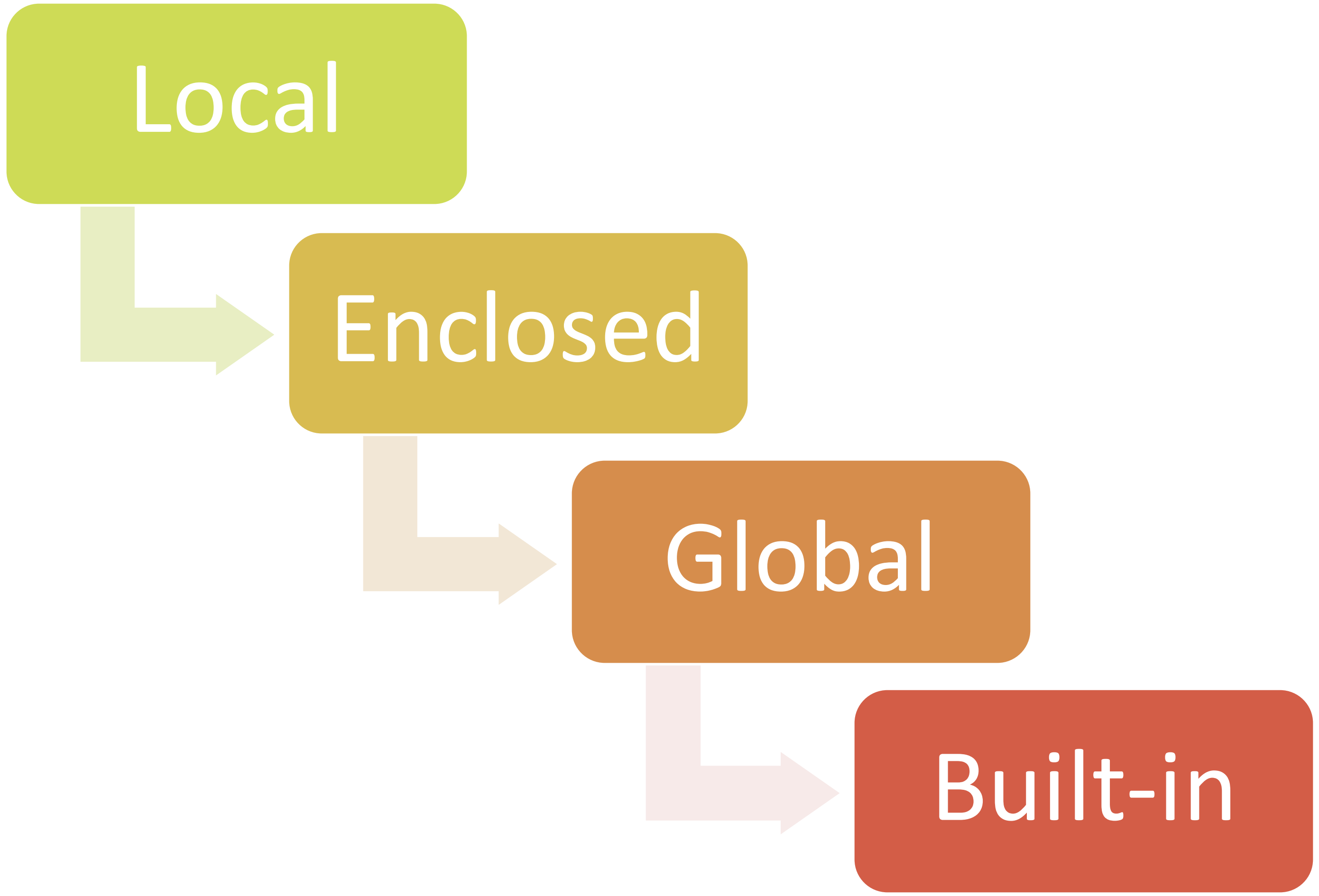
## Scopes

```
total = 0  # global

def sum(arg1, arg2):
    """Sum the parameters and return the result."""
    result = (arg1 + arg2) * total  # local
    print(f"Inside the function: {result}")

sum(10, 20)  # 0
```

## LEGB rule



https://sebastianraschka.com/Articles/2014_python_scope_and_namespaces.html

## Modifying the Behavior of a Python Scope

global

nonlocal

*https://realpython.com/python-scope-legb-rule/#modifying-the-behavior-of-a-python-scope*

## locals() and globals()

locals

globals

*https://realpython.com/python-scope-legb-rule/#using-scope-related-built-in-functions*

Thanks for attention