

Introduction to Python

T5 Bootcamp by SDAIA



SDAIA

الهيئة السعودية للبيانات
والذكاء الاصطناعي
Saudi Data & AI Authority

Functions

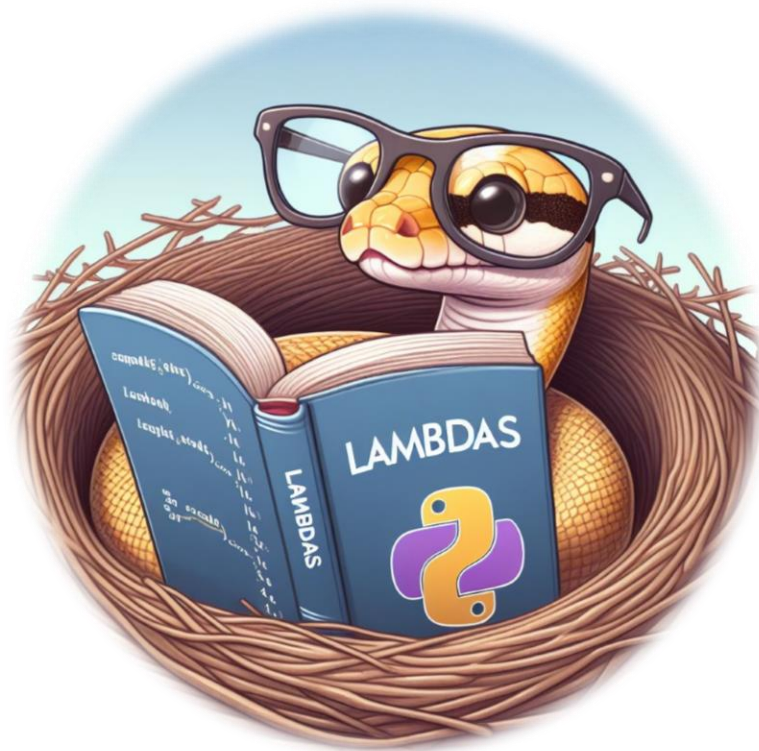
Modular Programming



SDAIA
الهيئة السعودية للبيانات
والذكاء الاصطناعي
Saudi Data & AI Authority

▶ Outline

- User-defined functions
- Naming Conventions
 - Reserved Keywords
- Docstrings
- Default Arguments
- Positional and Keyword Arguments
- Type Annotations
- Variables Scope
- Modules and Packages
- Call Stack
- Map
- Lambda Function



► Functions

- **Functions** are reusable blocks of code
- Python is built with standard library of functions:
 - Example: `sum`, `min`, `max`, `print` or `input` or `int` or `math`
- We can also define our own functions and give them names



► User-defined Functions

```
def add(num1, num2):  
    result = num1 +  
    num2  
    return result  
add(10, 20)
```

- Function name
- Variables in the definition are called parameters.
- We call functions and pass values; known as: arguments.
- Return value; Every function implicitly contains a return None statement at the end unless you have written your own return statement.



▶ Naming Conventions

- Be careful **not** to **pick taken names** such as: `sum`, `min`, `max`, `print` or `input` or `int` or `math`.
- Python will not prevent this behavior, so be careful! You will only see unexpected errors.



Reserved Keywords

Category	Keyword	Description
Logical operators	and, or, not, is	Operators used to combine Boolean expressions.
Conditional statements	if, elif, else	Keywords used to create conditional statements.
Loops	while, for, break, continue	Keywords used to create loops.
Exception handling	try, except, finally	Keywords used to handle exceptions.
Functions and classes	def, class, return, yield, lambda	Keywords used to define functions and classes.
Data types	None, True, False	Keywords used to represent special data types.
Other	import, from, as, assert, global, nonlocal, with, pass	Other keywords used for various purposes.



▶ Python docstrings

```
def bmi_calc(weight, height):  
    """Calculates the BMI  
  
    Args:  
        weight (float): weight in kg  
        height (float): height in meters  
  
    Returns:  
        float: kg/m^2  
    """  
    return weight / (height ** 2)
```

- **Docstrings** are essential in any mature library.
- Describe function purpose, parameters and return types.
- Docs help your memory, and your teammates, including the AI Copilot!



► Default Arguments

```
def greet(name, age=99):  
    print("Hello", name, age)
```

```
greet("John")  
greet("Mary", age=32)  
greet("Mary", 42)
```

- **name** is a positional argument; meaning that its position is important.
- Unlike **age**, which is a keyword argument.



► Python Type Annotations

- We can also add **Type Annotations** to function signatures:

```
def add(num1: int, num2: int) -> int:  
    z = num1 + num2  
    return z
```

Why Type Annotations?

- **Static type checkers** (like MyPy) can analyze type hints to **catch type errors early in development**.
- **IDEs** use type hints for better **autocomplete**.
- They act as part of the **function interface documentation** (which contribute to better developer experience)



► Type Annotations for Container Types

List of strings

```
my_fruits = ["apple", "banana", "orange"]
```

```
my_fruits: list[str] = ["apple", "banana", "orange"]
```

List of mixed types

```
mixed_data = [10, "hello", True]
```

```
mixed_data: list = [10, "hello", True]
```

Tuple of integers

```
product_dimensions = (10, 20, 30)
```

```
product_dimensions: tuple[int, int, int] = (10, 20, 30)
```



▶ Type Annotations for Container Types

```
# Set of unique colors
unique_colors = {"red", "green", "blue"}
unique_colors: set[str] = {"red", "green", "blue"}
```

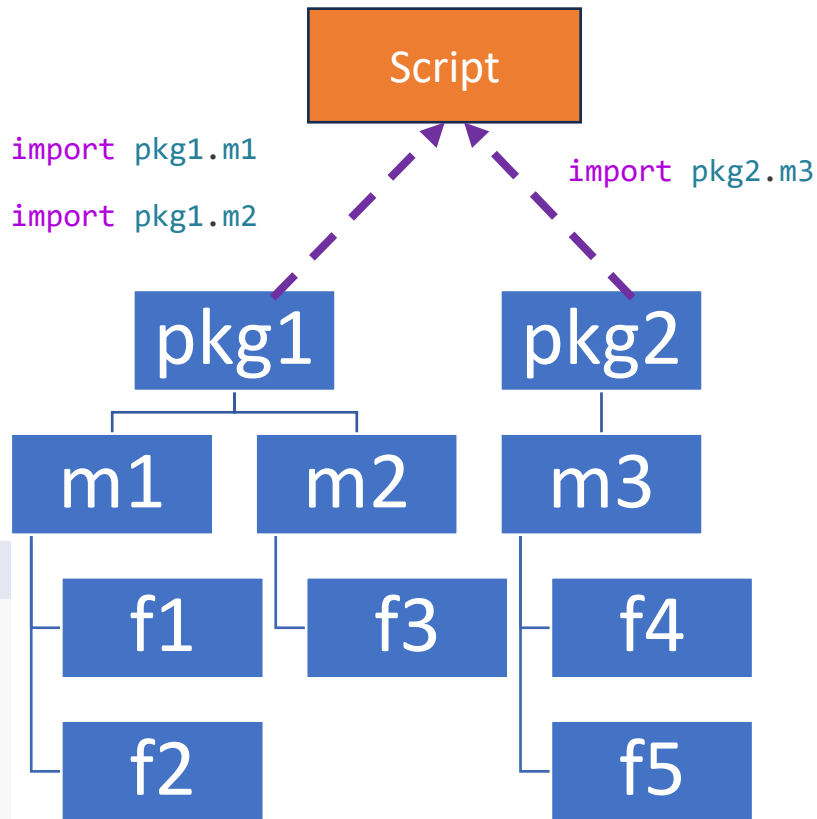
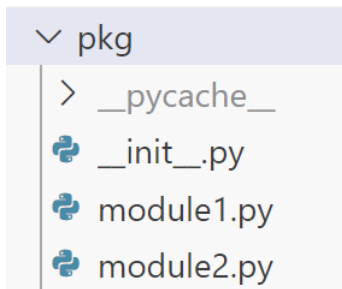
```
# Dictionary with string keys and integer values
customer_info = {"name": "John Doe", "age": 30}
customer_info: dict[str, int] = {"name": "John Doe", "age": 30}
```



▶ Script, Module, and Package

- **Function:** a reusable, callable block of code
- **Script:** (.py) file intended to be run directly
- **Module:** (.py) files intended to be imported into scripts and other modules
- **Package:** (folder) collection of related modules that aim to achieve a common goal

Note: a module can be written in **C** and loaded dynamically at run-time, like the re (regular expression) module.

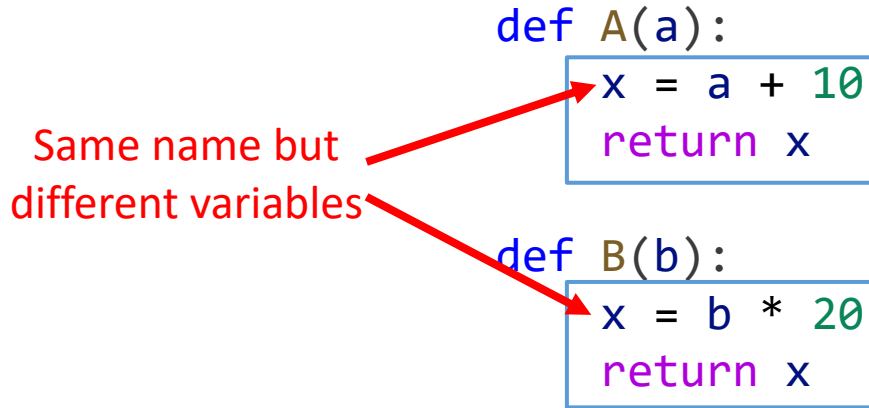


▶ Variable Scope

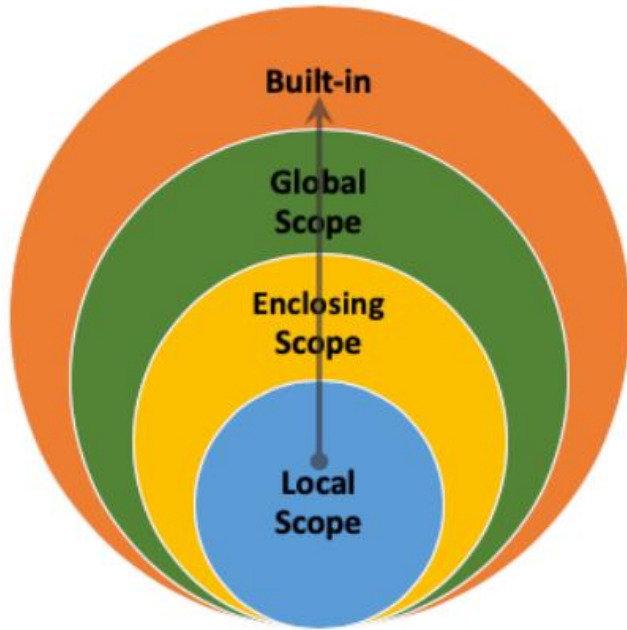
- Here, the variables `x` and `x` have the same name, but they have no relation whatsoever to each other, because they are in different **scopes**.

Same name but
different variables

```
def A(a):  
    x = a + 10  
    return x  
  
def B(b):  
    x = b * 20  
    return x
```



▶ Variable Scope



```
# Built-in scope  
print(x)
```

```
# Global scope  
x = 0
```

```
def outer():
```

```
    # Enclosed scope  
    x = 1
```

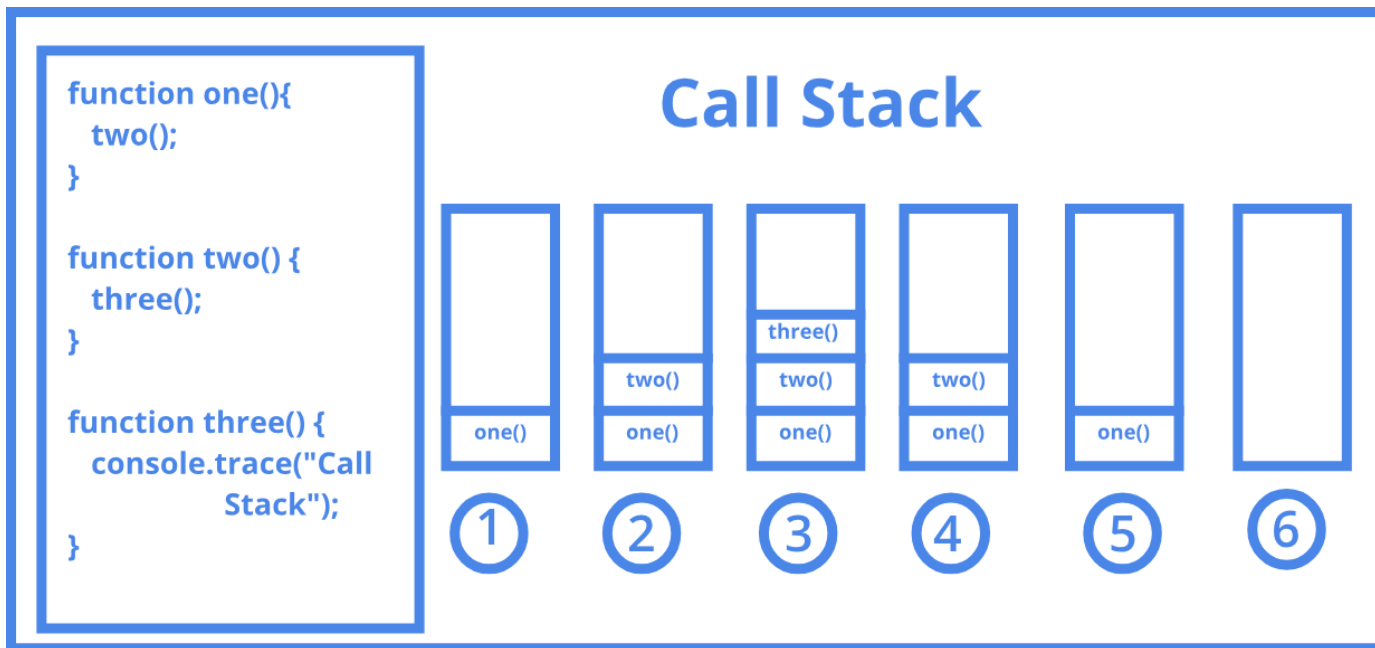
```
    def inner():
```

```
        # Local scope  
        x = 2
```



► Call Stack

- Functions can call other functions
- A call stack shows the trace of function calls



► To function or not to function? that is the question

- **Repeating:** If you find yourself repeating something, maybe you should wrap it up in a function.
- **Complex:** If you find testing a piece of code challenging, wrap it up in a function, and call it with various arguments to test it.
- **Composable:** If a block of code lends itself to be named and reused, it definitely should be a function.



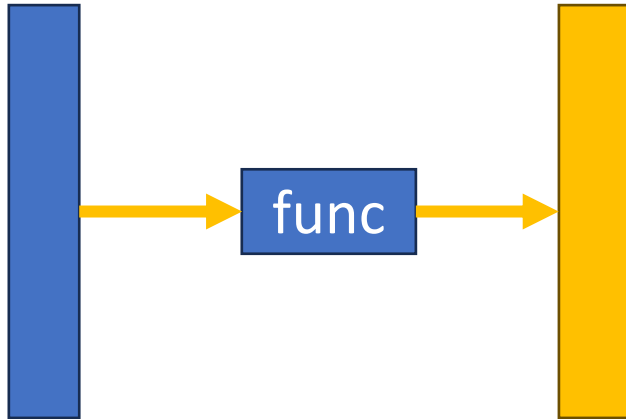
► Modular Programming

- Big programs are hard to work with.
- We break code into functions and classes.
- Modular code divides programs into logical parts.
- Each module focuses on one problem.
- Teams can work together better.
- This makes code easier to understand.



Map Function

- **map**: make an iterator that computes the function using arguments from each of the iterables.



```
def square(x):  
    return x * x
```

```
map(square, range(5))  
>> <map at 0x21ba1fa0640>
```

```
# iterate over the map object  
for x in map(square, range(5)):  
    print(x, end=' ')  
>> 0 1 4 9 16
```

```
# or convert it to list  
list(map(square, range(5)))  
>> [0, 1, 4, 9, 16]
```



Lambda Function

- A lambda function is a small anonymous function.
- A lambda function can take any number of arguments, but can only have one expression.
- Example sorting a list of tuples by the second key.
- The sort function takes a function that returns the sorting key.

```
def square(x):  
    return x * x  
  
square2 = lambda x: x * x
```

```
tuples = [('A', 3), ('B', 1),  
          ('C', 2), ('D', 4)]  
# sort items by 2nd element  
tuples.sort(key=lambda x: x[1])  
print(tuples)
```





Thank you



SDAIA
الهيئة السعودية للبيانات
والذكاء الاصطناعي
Saudi Data & AI Authority