Lesson 04 Summary: Functions

Data Science with Python – Key Concepts

Data Science Program

Functions Overview

Functions Summary

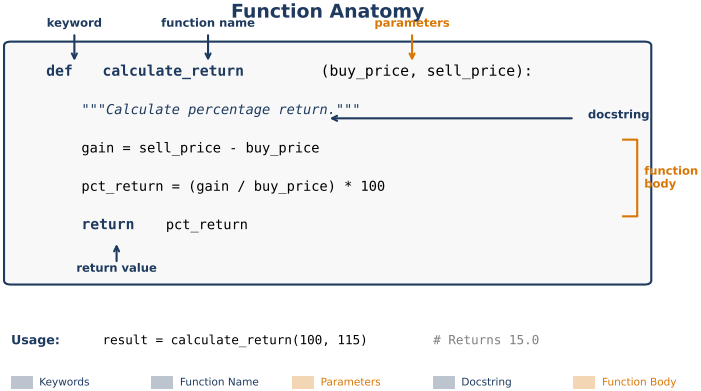
Define	Parameters	Return	Scope
`def func():`	`func(x, y=1)`	`return value`	`local/global`
Create reusable code	Input values	Output result	Variable lifetime

Function Template:

```
def function_name(param1, param2=default):
    """Docstring: what it does"""
    return result
```

Functions are reusable blocks of code with inputs and outputs

Function Anatomy



def keyword, name, parameters, colon, indented body, return

Parameters – Key Concepts

Four ways to pass data to functions:

- **Positional:** `func(a, b)` – order matters
- **Keyword:** `func(x=1, y=2)` – explicit naming
- **Default:** `def f(x=10)` – optional with default
- **Variable:** `*args, **kwargs` – flexible inputs

Example:

```
def calculate(price, days=252, include_fees=True):
    # days has default, price is required
```

Python passes object references – mutable objects can change

Return Values – Key Concepts

Sending data back from functions:

- **Single value:** `return price * 1.05`
- **Multiple values:** `return mean, std` (tuple)
- **No return:** Function returns `None`
- **Early return:** Exit function early with `return`

Unpacking multiple returns:

```
mean, std = calculate_stats(prices)
```

Functions without return statement return None

Variable Scope

Where variables exist and are accessible:

Local Scope:

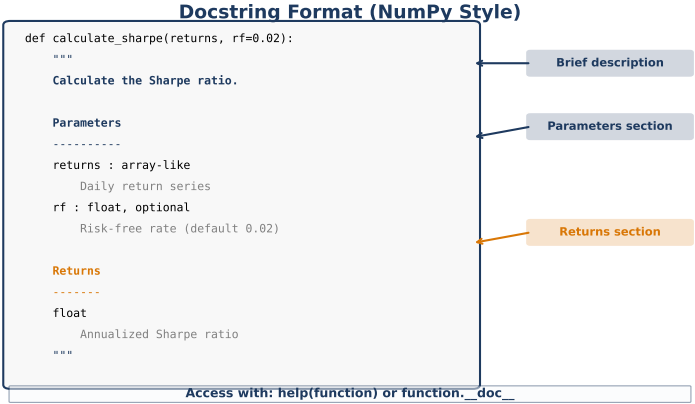
- Variables created inside function
- Destroyed when function ends
- Not accessible outside function

Global Scope:

- Variables created at module level
- Accessible everywhere (can read)
- Use `global` to modify (avoid!)

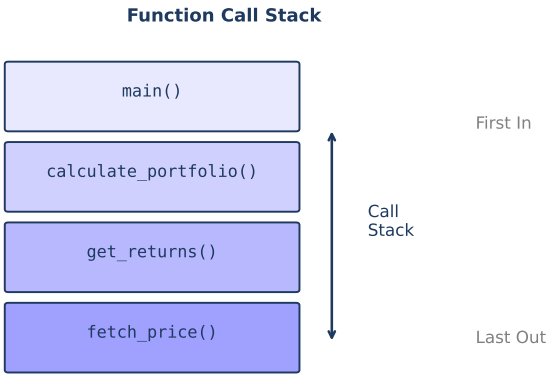
Local variables are destroyed when function ends

Docstrings



Access docstring with help(function) or function.__doc__

Call Stack



Stack grows with nested calls, shrinks as functions return

Pure vs Impure Functions

Pure Functions (preferred):

- Same input always gives same output
- No side effects (no external changes)
- Example: `def add(a, b): return a + b`

Impure Functions (use carefully):

- May modify external state
- May have different outputs for same input
- Example: `def update(lst, x): lst.append(x)`

Pure functions are easier to test and debug

Finance Functions Library

Essential Finance Functions

`calculate_return(p1, p2)`	Price change %
`annualize_return(daily_ret)`	Convert to yearly
`calculate_volatility(returns)`	Standard deviation
`sharpe_ratio(ret, rf)`	Risk-adjusted return
`max_drawdown(prices)`	Largest peak-to-trough
`beta(stock, market)`	Market sensitivity

Build your own library of financial calculation functions

Quick Reference

Essential Syntax:

Concept	Syntax
Define function	`def func_name(params):`
Return value	`return value`
Default param	`def f(x=10):`
Multiple returns	`return a, b`
Unpack returns	`x, y = func()`
Docstring	`"""Description"""`
Access docs	`help(func)`

Best Practice: Keep functions small, single-purpose, and pure

Master functions for modular, testable code