Python functions

More on Defining Functions

JINU KURIEN DANIEL

1: Introduction to Special Parameters in Python Functions

- **Definition**: Special parameters control how arguments are passed to functions in Python.

- Types:

    - Positional-only (/)
    - Keyword-only (*)
    - Positional-or-Keyword (no marker)

2: Why Use Special Parameters?

- **Readability**: Clarifies how to call the function.
- **Safety**: Prevents misuse or ambiguity in function calls.
- **Performance**: Efficient handling of arguments internally.

3: Syntax of Special Parameters

*def example(pos_only1, pos_only2, /, pos_or_kwd, *, kwd_only1, kwd_only2):*

- **Positional-only parameters (/)** must be passed positionally.
- **Keyword-only parameters (*)** must be passed explicitly by keyword.
- Parameters without markers can be either positional or keyword.

4: Positional-or-Keyword Arguments

- Default behavior if no markers (/ or *) are used.

```
def standard_arg(arg):
    print(arg)

standard_arg(2)      # positional call
standard_arg(arg=2)  # keyword call
```

5: Positional-only Parameters (/)

- Arguments must be passed by position, not keyword.

```
def pos_only_arg(arg, /):
    print(arg)

pos_only_arg(1)       # correct
pos_only_arg(arg=1)   # TypeError
```

6: Keyword-only Parameters (*)

- Arguments must be explicitly named (passed by keyword).

```
def kwd_only_arg(*, arg):
    print(arg)

kwd_only_arg(arg=3)  # correct
kwd_only_arg(3)      # TypeError
```

7: Combined Example (Positional, Positional-or-Keyword, Keyword-only)

```
def cmb_sample(pos_only, /, std, *, kwd_only):
    print(pos_only, std, kwd_only)

cmb_sample(1, 2, kwd_only=3)      # correct
cmb_sample(1, std=2, kwd_only=3)  # correct
cmb_sample(pos_only=1, std=2, kwd_only=3)  # TypeError
```

10: Arbitrary Argument Lists (*args)

- Capture multiple positional arguments into a tuple:

```
def concat(*args, sep="/"):
    return sep.join(args)

concat("earth", "mars", "venus")         # 'earth/mars/venus'
concat("earth", "mars", "venus", sep=".") # 'earth.mars.venus'
```

12: Lambda Expressions (Anonymous Functions)

- small anonymous functions
- single line code
- Syntax

lambda arguments: expression

```
sum_ab = lambda a, b: a + b

print(sum_ab(3,5))
```

13: Documentation Strings (Docstrings)

- Explain clearly what functions do.
- ‘’’Short summary line + blank line + detailed description’’’

```
def my_function():
    """Do nothing, but document it.

    This function literally doesn't do anything.
    """
```

- Access with __doc__:

```
print(my_function.__doc__)
```

14: Function Annotations (Optional Metadata)

- notes or hints about the types
- optional - not affect the code
- stored in directory __annotations__
- better readability
- used for tools like mypy(type checker),

```
def sum(a : int , b : int) -> int:
    return a + b
print("Function Annotations = " , sum.__annotations__)
```

15: Python Coding Style (PEP 8 - Style Guide - Highlights)

- **Indentation**: 4 spaces (no tabs).
- **Line length**: ≤79 characters.
- **Blank lines**: separate logical sections.
- **Whitespace**: use spaces around operators and commas.
- **Naming conventions**:
  - Classes: UpperCamelCase
  - Functions and methods: lowercase_with_underscores
- Use clear and descriptive names, maintain readability.

Q & A

Thank You