CoGrammar

Elevating Your Python Functions Best Practices & Documentation Pt. 1

SKILLS FOR LIFE

SKILLS BOOTCAMPS

Department for Education

Lecture Objectives

1. **Explore additional best practices tailored for functional programming**

Poll:

Assessment

CoGrammar

Recap on Previous Week

Best Best Practices

★ **Descriptive Function Names:**
Instead of foo() or bar(), let's name our functions so that anyone reading our code knows exactly what's going on.

```
def calculate_area(radius):
    # Code for calculating area
```

★ **Single Responsibility Principle:**
One function, one responsibility. Break down your code into smaller, focused functions.

```
# Step 1:
def fetch_data(url):
    print("Code to fetch data from the URL.")

# Step 2:
def process_data(data):
    print("Code to process this data.")
```

★ **Default Argument Values:**

Be careful with default values. They're handy but watch out for mutable defaults.

```
def add_item(item, items=[]):
    items.append(item)
    return items

# Better:
def add_item(item, items=None):
    if items is None:
        items = []
    items.append(item)
    return items
```

★ **Avoiding Global Variables:**

Global variables can be tricky. Stick to local scope and keep your functions pure.

```
count = 0
def increment_count():
    global count
    count += 1
    return count

#   Better:
def increment_count(count):
    return count + 1
```

Input Validation and Error Handling

★ **Input Sanity Checks:**
Input validation is like checking your ingredients before cooking. Ensure your functions get what they expect.

```
def calculate_area(radius):
    if not isinstance(radius, (int, float)):
        raise ValueError("Radius must be a number")
    # Code for calculating area
```

★ **Exception Handling:**
Expect the unexpected. Wrap your code in try-except blocks and provide meaningful error messages.

```
def divide_numbers(a, b):
    try:
        result = a / b
    except ZeroDivisionError:
        raise ValueError("Cannot divide by zero")
    return result
```

★ **Logging:**
Logging is your friend. Add informative logs to aid debugging.

```
import logging

def process_data(data):
    logging.info(f"Processing data: {data}")
    # Code to process the data
```

Poll:

Assessment

Wrapping Up

Best Practices

In conclusion, embracing best practices in your Python functions not only enhances the clarity and maintainability of your code but also elevates you as a conscientious and effective developer.

Input Validation & Error Handling

To sum it up, thorough input validation not only safeguards your functions from unexpected errors but also contributes to the overall robustness and reliability of your Python applications.

CoGrammar

Questions around Best Practices

CoGrammar

Thank you for joining