Welcome to this CoGrammar Lecture:

Functions

The session will start shortly...

Questions? Drop them in the chat.
We'll have dedicated moderators
answering questions.

Software Engineering Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

Software Engineering Session Housekeeping cont.

- For all **non-academic questions**, please submit a query: **www.hyperiondev.com/support**
- Report a **safeguarding** incident: **www.hyperiondev.com/safeguardreporting**
- We would love your **feedback** on lectures: **Feedback on Lectures**

Skills Bootcamp
8-Week Progression Overview

Fulfil 4 Criteria to Graduation

✅ Criterion 1: Initial Requirements

Timeframe: First 2 Weeks
Guided Learning Hours (GLH): Minimum of 15 hours
Task Completion: First four tasks

Due Date: 24 March 2024

✅ Criterion 2: Mid-Course Progress

60 Guided Learning Hours

Data Science - **13 tasks**
Software Engineering - **13 tasks**
Web Development - **13 tasks**

Due Date: 28 April 2024

CoGrammar

Skills Bootcamp Progression Overview

✅ Criterion 3: Course Progress

Completion: All mandatory tasks, including Build Your Brand and resubmissions by study period end
Interview Invitation: Within 4 weeks post-course
Guided Learning Hours: Minimum of 112 hours by support end date (10.5 hours average, each week)

✅ Criterion 4: Demonstrating Employability

Final Job or Apprenticeship Outcome: Document within 12 weeks post-graduation
Relevance: Progression to employment or related opportunity


SKILLS FOR LIFE
SKILLS BOOTCAMPS
Department for Education
CoGrammar
Functions
March 2024

Agenda

- Functions
- Scope
- Stack Traces
- Debugging

Functions

CoGrammar

What is a function ?

- A function is a fundamental building block of programming that encapsulates a set of instructions or operations designed to perform a specific task. In essence, it's a named block of code that can be invoked (called) multiple times throughout a program, allowing for code reuse, modularity, and abstraction.

Function Syntax

```
def function_name(parameters):
    """docstring"""
    # function body
    return value
```

- **Function Name:** A unique identifier for the function.
- **Parameters (Arguments):** Inputs to the function, specified within parentheses.
- **Function Body:** The block of code that performs the desired task.
- **Return Statement:** Optional statement that specifies the value returned by the function.

- To execute a function, you call it by using its name followed by parentheses, optionally passing arguments.

```
function_name(arugments) # without return type

func_output = function_name(arguments) # with return type
# storing function result in variable 'func_output'
```

Functions

Parameters, Arguments &
Return Values

CoGrammar

Parameters VS Arguments

Parameters are variables defined in the function header.

Arguments are values passed to the function when it is called.

```
def greet(name):  # 'name' is a parameter
    """This function greets the user by name."""
    print(f"Hello, {name}!")

# Calling the function with an argument
greet("Lord Voldemort")  # "Lord Voldemort" is an argument
```

Return Statement

- In Python, the `**return**` statement is like sending a message back to where you called a function from. It's a way for the function to finish its job and share its final answer with the rest of the program.

Imagine you ask a friend to solve a math problem for you. After working on it, your friend comes back to you with the solution. In Python, the `return` statement is like your friend giving you that solution. It's the way the function tells the rest of the program what answer it found.

Function with Return Value

```python
def add(a, b):
    """This function adds two numbers and returns the result."""
    return a + b

sum_one = add(2, 4)
print(sum_one)    # Output: 6

# Printing Result
print(add(2, 3))    # Output 5
```

The **add()** function takes two parameters **a** and **b**, adds them together, and returns the result.

Function Types

CoGrammar

Function Types

Built-in functions

Built-in functions in Python are like special tools that Python already knows how to use. They're like ready-made functions that you can use anytime without having to write the code for them yourself.

User-defined functions

User-defined functions, on the other hand, are functions that you create yourself. They're like custom-made tools that you design for specific tasks. With user-defined functions, you get to write the code for what you want the function to do.

Built-in Functions

- These functions are built into Python, so you can use them right away without needing to define them first.
- Examples of built-in functions include:
  - print()
  - len()
  - input()
  - max()
  - min()

User-Defined Functions

- You define these functions by writing their names, specifying what inputs they need (if any), and describing what they should do with those inputs.

```
def general_greet(name):
    """This function greets the user."""
    print(f"Well Hello There {name}!")

general_greet("Pikachu")
# Output: Well Hello There Pikachu!
```

Scope

CoGrammar

What is a Scope ?

- **Scope** refers to the accessibility of variables in different parts of a program. Variables defined within a function are typically scoped to that function, meaning they can only be accessed within that function.

What is a Scope ?

There are different types of scope:

- **Global scope:** Variables defined outside of any function, accessible throughout the entire program.
- **Local scope:** Variables defined within a function, only accessible within that function.
- **Block scope:** Some languages have block scope, where variables defined within a block (e.g., within an `if` statement or a loop) are only accessible within that block.

Global Scope

- Variables and functions defined outside of any function or class have global scope. They can be accessed from anywhere in the program, including inside functions and classes.

```
x = 10  # Global variable

def my_function():
    print(x)  # Accessing global variable 'x' inside a function

my_function()  # Output: 10
```

Local Scope

- Variables and functions defined inside a function have local scope. They can only be accessed from within that function and are not visible outside of it.

```
def my_function():
    y = 20  # Local variable
    print(y)  # Accessing local variable 'y' inside the function

my_function()  # Output: 20
```

Stack-Traces

Stack traces are automatically generated when an error occurs in Python. They show the sequence of function calls leading up to the error.

```python
def divide(a, b):
    return a / b

def main():
    result = divide(10, 0)

main()
```

```
Traceback (most recent call last):
  File "example.py", line 7, in <module>
    main()
  File "example.py", line 5, in main
    result = divide(10, 0)
  File "example.py", line 2, in divide
    return a / b
ZeroDivisionError: division by zero
```

What is a Scope ?

When an error occurs, Python captures the current state of the program's execution stack and prints a stack trace to the console. This stack trace includes:

- **Error Type:** The type of error that occurred (e.g., SyntaxError, NameError, TypeError).
- **Error Message:** A description of the error, providing additional details about what went wrong.
- **Traceback:** A list of function calls, starting from the point where the error occurred and going back to the initial entry point of the program.

Debugging

CoGrammar

Debugging in Python

- **Debugging** is the process of identifying and fixing errors or bugs in a program. It involves analyzing the behavior of the code to understand why it is not working as expected and making the necessary corrections to resolve the issues.
- **Debugging** is an essential skill for programmers, and mastering it can greatly improve your ability to write reliable and efficient code.

Steps in Debugging

- **Reproduce the Issue:** Start by reproducing the problem. Run the program and observe the behavior that leads to the error or unexpected output.
- **Examine Error Messages:** Pay attention to any error messages or exceptions that are raised. These messages often provide valuable information about what went wrong and where the problem occurred.
- **Use Print Statements:** Insert print statements at different points in the code to track the values of variables and understand the flow of execution. Print statements can help you identify which parts of the code are executing and what values are being used.
- **Inspect Data:** Check the input data and intermediate values to ensure they are as expected. Verify that variables contain the correct data and are being manipulated correctly.

Steps in Debugging cont.

- **Use Debugging Tools:** Python provides debugging tools in the integrated development environments (IDEs) such as PyCharm, VS Code, and Jupyter Notebooks, which offer features like breakpoints, step-through debugging, and variable inspection to help you analyze and troubleshoot code more effectively.
- **Isolate the Problem:** Narrow down the scope of the problem by identifying the specific section of code where the error occurs. Focus on isolating the root cause of the issue rather than trying to fix everything at once.
- **Fix the Issue:** Once you have identified the problem, make the necessary corrections to fix it. This may involve rewriting code, adjusting logic, or updating data structures to resolve the issue.
- **Test the Fix:** After making changes, test the program again to verify that the issue has been resolved. Ensure that the program behaves as expected and that the error no longer occurs.
- **Document Changes:** Document any changes made during the debugging process, including the steps taken to identify and fix the problem. This documentation can be helpful for future reference and for communicating with other developers.

Summary

- Debugging plays a vital role in software development, helping programmers swiftly spot and fix errors in their code. By tackling bugs head-on, debugging not only boosts the quality and dependability of software but also saves time that would otherwise be wasted on manual error hunting.
- It promotes a better grasp of code functions and logic, aids in performance fine-tuning, and guarantees the accuracy and resilience of programs.

Let’s take a break

CoGrammar

Questions and Answers

CoGrammar

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar