CoGrammar

Defensive Programming

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

Data Science Lecture Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(FBV: Mutual Respect.)**
- 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 Open Classes. You can submit these questions here: **Open Class Questions**

Data Science Lecture 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**

Lecture Objectives

- **Discover the different types of errors that could occur in your programs and how to handle them.**
- **Understanding exceptions and how to create them.**

Everyone Makes Mistakes

- No programmer is perfect, and we're going to make a lot of mistakes in our journey - and that is perfectly okay!
- What separates the good programmers from the average ones is the ability to find and debug errors they may encounter.

Error Messages

- The output window of our IDE will usually show any and all error messages if an error or mistake is detected.
- It should display the type of error found as well as the line number in your code where the error occurred.
- Your program will stop running immediately when an error occurs.

Error Message Example

```
Traceback (most recent call last):
  File "C:/Users/[redacted]/I AM A PYTHON FILE.py", line 9, in <module>
    print(name + " is " + age + " years old" )
TypeError: can only concatenate str (not "int") to str
```

- ★ **Looking at the above example :**
  - **The message states that the error happened at line 9 - a good.**
  - **It also states the type of error, which appears to be a TypeError. Useful, as this already provides some idea how to fix the error.**

Syntax Errors

- Some of the easiest errors to fix...
  - ... Usually
- Mainly caused by typos in code or Python specific keywords that were misspelled or rules that were not followed.
- When incorrect syntax is detected, Python will stop running and display and error message.

Syntax Error Example

```
user_input = input("enter name : "
# input missing closing brackets

print("Hello World!)
# Missing quotation mark

age = 2022 - date_of_birth
print(dat_of_birth)
# Misspelled variable name
```

Indentation Errors

- Indentation is important in programming.
- Python uses indentation to understand where blocks of code start and stop.
- The presence of indentation errors means that there is something wrong with the structure of the code.
- A good rule of thumb: if a line of code ends with a colon (:), the next line should be indented.

Indentation Error Example

```
cold = False

if cold :

print("Wear a jacket!")
# Indentation error, print statement is meant to
#   be within the if statement.
```

Type Errors

- A type error occurs when your code has misinterpreted one type of data for another, like integers for strings.
- Remember that for Python to actually work, your code needs to make logical sense so that Python can interpret it correctly and achieve the desired output.

Type Error Example

```
maths = "Sixty" * "Seven"
# Type error, python cannot multiply strings together.


temperature = "26 degrees" > 21
# Type error, cannot use logical operators to compare
#   string to int

# Type errors occur when Python cannot interpret
#   something that makes no logical sense.
```

Name Errors

- Naming errors occur when you try to reference or call a variable that has not been declared / created yet.
- A good habit to into when coding is to first define all variables, functions, etc. at the beginning of our programs.

Name Error Example

```
print("Welcome " + user + ", please make a selection.")
user = input("Enter your user name : ")

# Name error, user referenced before declaration.
```

Logical Errors

- Logical errors occur when your program is running, but the output we are receiving is not what we expected.
- The code could be typed incorrectly, or perhaps an important line has been omitted, or the instructions given to the program has been coded in the incorrect order.

Logical Error Example

```
years_old = "32"

months_old = years_old * 12

print("If you are " + str(years_old) + " years_old, you are " +
      str(months_old) + " months old!" )

# The code runs, however there is a flaw in the logic.
#   The value of months_old is printed 12 times, instead
#       of the number of months.
#           This is because year_old is a string, not integer.
```

Let’s Breathe

Let’s take a small break before moving on to the next topic.

Try - except Blocks

- ★ We should try to code in a way where we can anticipate potential errors. Such as :
  - User errors
  - Environment errors
  - Logical errors
- ★ We can write code to ensure that these types of errors do not crash our code base.
- ★ We have two ways of doing this - if statements and try-except blocks.

Handling Errors

- **If statements**
  - **Easy way of anticipating errors - if input is not correct, we can use control structures to potentially correct it.**
  - **For example, if our user is trying to register a username that already exists, we can prompt the user for a new username.**

Example

```
usernames = ["Jimmy", "Billy", "John"]

while True:

    username = input("Please enter your username : ")
    password = input("Please enter your password : ")

    if username in usernames :

        print("This username already exists! Try again.")
        continue

    else:

        print(f"Welcome {username}.")
        break
```

Try - Except Block

- When handling exceptions, there are 2 main parts.
  - The try section, which is the part that we are attempting to execute if no exceptions occur.
  - The except section, which is the part that will execute when an exception does occur.

Example

```
while True:

    try:

        num = int(input("Please enter a number : "))

        print("That a nice number right there.")
        break

    except ValueError:
        print("That is not a valid number. Please try again. . .")
```

Raising Exceptions

- There will be occasions when you want your program to raise a custom exception whenever a certain condition is met.

- In Python we can do this by using the “raise” keyword and adding a custom message to the exception: In the below example we are asking the user to input a value greater than 10. If the user enters a number that does not meet that condition, an exception is raised with a custom error message.

Example

```
num = int(input("Please enter a value greater than 10 : "))

if num < 10:

    raise Exception(f"Your value was less than 10. The value of num was : {num}")
```

A Note on try-except

- ★ It may be tempting to wrap all code in a try-except block. However, you want to handle different errors differently.
- ★ Don’t try to use try-except blocks to avoid writing code that properly validates inputs.

CoGrammar

Q & A SECTION

Please use this time to ask any questions relating to the topic, should you have any.

CoGrammar

Thank you for joining!