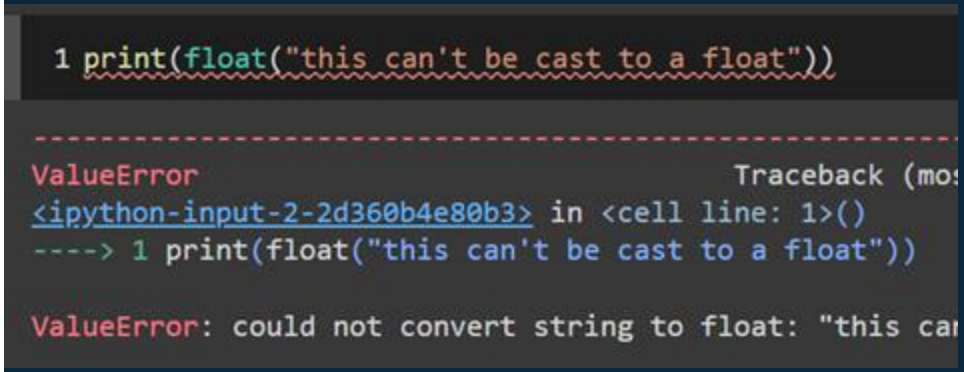
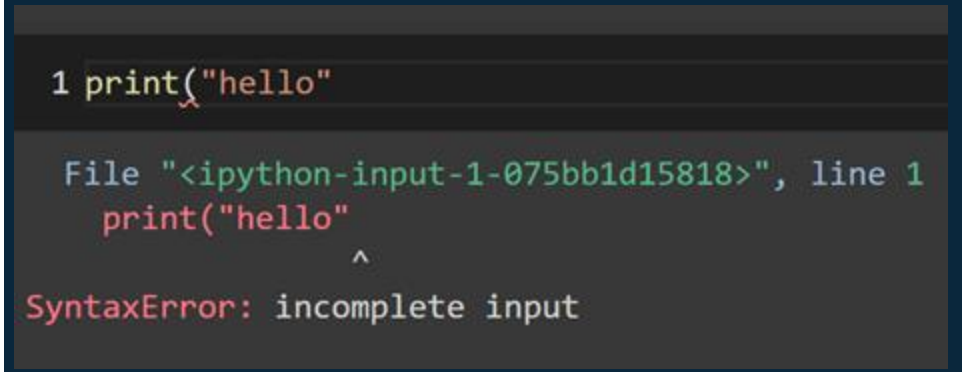
Exceptions

can code communities

Whenever you run your code and an error message shows up, this is called an exception.



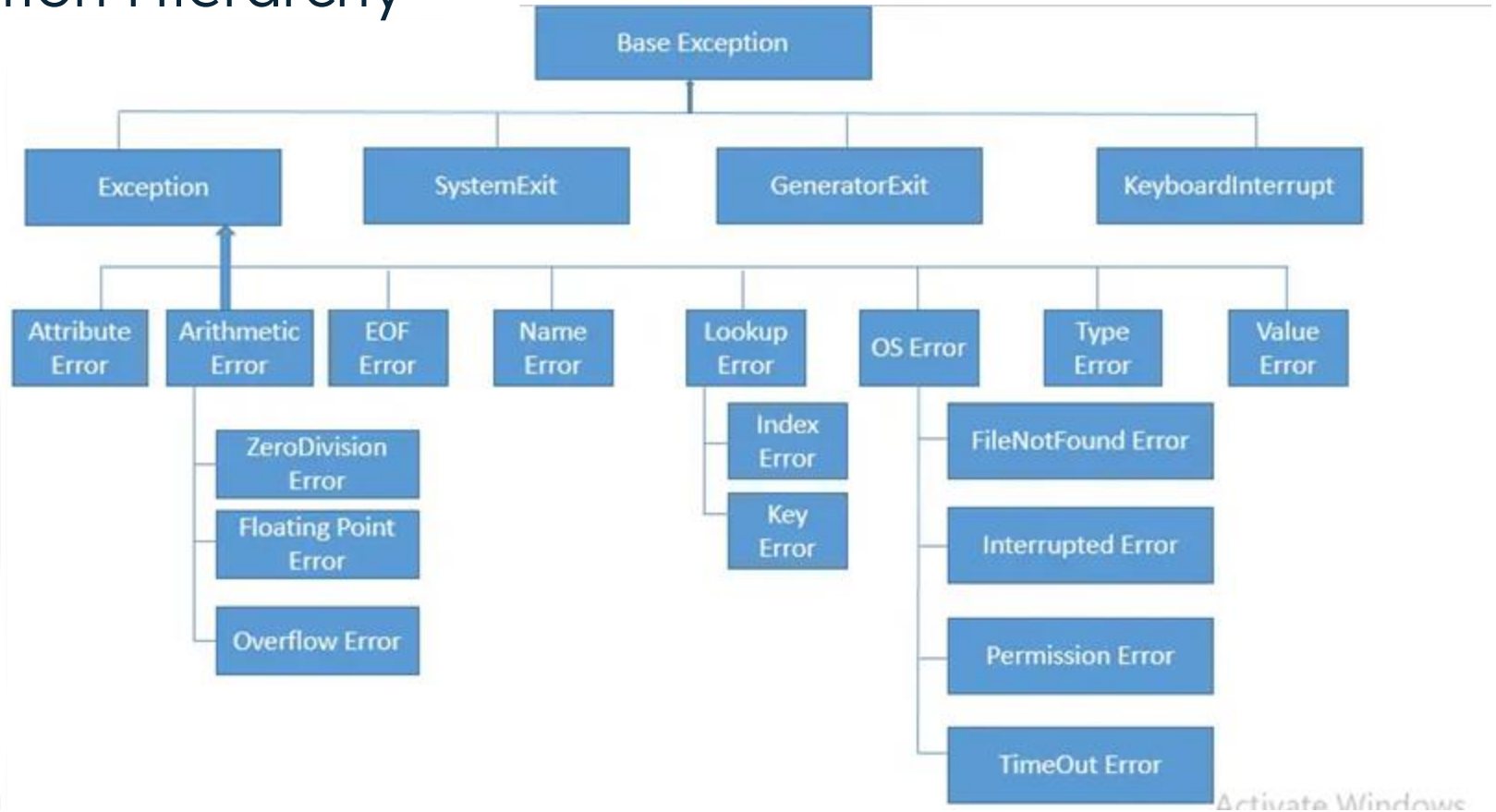


Compile-time vs Runtime Errors

There are two main types of errors in code: Compile-time and Runtime

- **Compile-time errors**, also called static errors, happen when you violate the rules of writing syntax. These usually have a red or yellow underline in your IDE, so you can fix them before you run your code.
    - These are more relevant in languages that are compiled. Python is interpreted, so there is no compiler, so technically there are no compile-time errors.
- **Runtime errors**, also called dynamic errors, happen when you run your code. It's harder to discover them. In Python, most exceptions are run-time errors.

Exception Hierarchy



Exception

- Exception is a class derived from BaseException
- There are various types of exceptions that will occur if your code has runtime errors, these are just a few common examples:
    - AttributeError - when an attribute or function not associated with a data type is referenced on it (ex: calling a method on an int)
    - NameError - when you try to use a variable, function, or module that doesn't exist (ex: using a variable or function yet to be defined)
    - ValueError - when a function is called with the proper argument type but with the wrong value (ex: casting a non-numerical string to a float)
    - TypeError - when an operation is performed on an incorrect data type (ex: using a list method on a dictionary)

KeyboardInterrupt

- The other class derived from BaseException is KeyboardInterrupt
- **This is not an error in your code** - it occurs if the user kills the program
    - For example, if your code stops to take user input, or enters an infinite loop, you can kill the program by clicking the stop sign or typing “CTRL + C”
- This produces the KeyboardInterrupt exception

Catching Exceptions

- You can use try/except statements to catch exceptions
- First, Python will try running the **try** block
  - If the **try** block would produce an exception, the **except** block gets run instead, and the try block is skipped
  - If the **try** block runs without exceptions, then the except block is skipped
- Put a specific name of an exception in the except block
  - This will catch that particular exception, **and** any exception classes derived from it (but not exception classes from which *it* is derived)

No ValueError - runs try block

```
1 userin = input("Enter a number: ")
2 num_list = []
3
4 try:
5     num_list.append(float(userin))
6 except ValueError:
7     print("Error: Must be a number")
8
9 print(num_list)
```

```
Enter a number: 5.5
[5.5]
```

ValueError - runs except block

```
1 userin = input("Enter a number: ")
2 num_list = []
3
4 try:
5     num_list.append(float(userin))
6 except ValueError:
7     print("Error: Must be a number")
8
9 print(num_list)
```

```
Enter a number: cat
Error: Must be a number
[]
```

Exercise - Handling Invalid User Input

Write a Python program that takes a customer's age as user input and determines whether they're eligible for a senior discount.

Sometimes the age might not be in the correct format. Handle this using `try-except`, and print a descriptive error message if the age can't be cast to an int.

If the age is greater than or equal to 65, the customer is eligible for the discount. Otherwise, they're not eligible. Print whether the customer is eligible or not.

Try/Except/Else

You can add an `else` block that will run if no exception occurs

```
try:
#Statements to try
except nameOfException:
#Statements to execute upon that exception
else:
#Statements when no exception occurs
```

Try/Finally

You can add a `finally` block that will be executed regardless if the try block raises an error. This is good for cleaning up resources, because it will **always** be run.

A more practical use of `try-finally` is making sure files are always closed.

```
def append_user_input(lis):
    try:
        # Tries to add a float to the list
        lis.append(float(input("Enter a number: ")))
    except ValueError:
        # What to do if the user input can't be cast to a float
        print("Invalid input")
        return None
    finally:
        # This gets run no matter what, even if
        #  the function returns in the except block
        print("Your list:", lis)

lis = []
append_user_input(lis)
```

Raising Exceptions

You can choose to raise your own exceptions, using this syntax:

```
raise ExceptionName("Error message")
```

This can be useful for catching unique exceptions that might cause errors in your program, but aren't necessarily caught by Python by default.

Make sure to use the specific exception type that fits your error the best. For example, if the user inputs the wrong data type, you can raise a ValueError.

Exercise - Raising Exceptions

Write a program to take the square root of user input.

Use a `try-except` statement to ensure the user inputs a float.

If the user inputs a negative number, **raise a ValueError** that will also be caught by the except statement. Make sure to write a descriptive message in the exception you raise.

Propagating Exceptions

If you raise an exception in a function, you can catch the exception when you call the function.

You can also choose to raise the exception again, propagating it as much as you want through your function calls.

Let's say you have a function called **`my_function`** that raises a ValueError in some cases:

```
try:
        my_function()
except ValueError:
        print("Calling this function caused a ValueError")
```

Exercise

You have been assigned the task of creating a sales tax calculator for an e-commerce company. Write a Python function called `calculate_final_price` that takes the price of a product and the sales tax rate, and return the final price including tax.

The price should be a positive number, and the tax rate should be between 0 and 1 (exclusive). If either of them are outside of the valid range, raise a custom `ValueError` with an appropriate error message.

Now, test your implementation by asking the user to input a product price and sales tax rate, and call your function. Catch any potential `ValueError` raised by the function.