

CS65: Introduction to Computer Science

Errors and Exceptions



Md Alimoor Reza
Assistant Professor of Computer Science

Announcement

- **Final Exam**

Week 16 (Tue: 05/10)	Final Exam Tuesday 5/10/2022 (9:30 AM - 11:20 AM) Collier-Scripps Hall, C-S 0301
----------------------	---

- **Quiz 6 (Last one)**

- Thursday 04/28/22
- Topics: Classes/Objects

- **Final Project Presentations**

- Tuesday: 05/03/22
- Thursday: 05/05/22
- Who wants to volunteer for Tuesday's presentation?

Topics

- **Exception:** action taken outside of the normal flow of control because of errors or unacceptable conditions

Exceptions

- **Errors that crash your program**
 - incorrect syntax
 - value issues
 - division with zero
- Avoiding these errors requires us to explicitly handle them
- **Try-except block:** Try some code, if an exception occurs then run except block instead of crashing
- Reference: <https://docs.python.org/3.4/tutorial/errors.html>

Exceptions

- **Errors that crash your program**
- **Try-except block:** Try some code, if an exception occurs then run except block instead of crashing
- **Two options:**
 - Continue executing the codes beyond the try-except block
 - Run a loop around it to allow the user to correct it

Try-Except Block

try:

<< code that might create an exception. In that case the execution of this code stops and jumps to the except block >>

except:

<< when an exception occurs, this code is executed >>

else:

<< code that runs when exception does not occur >>

finally:

<< code that always run regardless of the occurrence of an exception or not >>

Example: Without Try-Except Block

```
def main():  
    num = input("Enter a number: ")  
    num = float(num)  
    print("The square of the number user entered is: ", num*num)  
    return None  
  
main()
```

Enter a string instead of a number and see what happens

Example: Try-Except-Else Block

```
def main():  
    try:  
        num = input("Enter a number: ")  
        num = float(num)  
  
    except:  
        print("User failed to enter a number ... ")  
  
    else:  
        print("The square of the number is: ", num*num)  
  
    return None  
  
main()
```

Enter a string instead of a number and see what happens

Example: Try-Except-Else-Finally Block

```
1
2
3 def main():
4
5     # example
6     try:
7         num = float(input("Enter a number: "))
8         print("Use entered a number ... ")
9
10    except:
11
12        print("Use failed to enter a number ...")
13
14    else:
15
16        print("The square of the number user entered is: ", num*num)
17
18    finally:
19
20        print("Adding 10 to the square of the number: ", num*num + 10)
21
22
23 main()
```



User enters a string instead of a number

Example: Try-Except-Else-Finally Block

```
1
2
3 def main():
4
5     # example
6     try:
7         num = float(input("Enter a number: "))
8         print("Use entered a number ... ")
9
10    except:
11
12        print("Use failed to enter a number ...")
13
14    else:
15
16        print("The square of the number user entered is: ", num*num)
17
18    finally:
19
20        print("Adding 10 to the square of the number: ", num*num + 10)
21
22
23 main()
```



User enters a valid number

```
Enter a number: 10
Use entered a number ...
The square of the number user entered is: 100.0
Adding 10 to the square of the number: 110.0
```

Multiple Exceptions

- Different types of exceptions in Python

Exception Type	Description
IOError	I/O operation fails, e.g. trying to open a non-existent file *
IndexError	tried to index into a structure with a not-present index.
KeyError	tried to access non-existent key in a dictionary.
NameError	identifier for a name couldn't be found in scope.
SyntaxError	syntax error encountered.
TypeError	type error encountered, e.g. argument to built-in is of wrong type.
ValueError	built-in function/operation received value of right type, but wrong value (e.g., got a str, but it didn't represent a number)
ZeroDivisionError	tried to divide by zero.

Multiple Exceptions

- Different types of exceptions hierarchy in Python

```
BaseException
+-- KeyboardInterrupt
+-- Exception
    +-- ArithmeticError
    |   +-- ZeroDivisionError
    +-- EnvironmentError
    |   +-- IOError
    +-- EOFError
    +-- LookupError
    |   +-- IndexError
    |   +-- KeyError
    +-- NameError
    +-- SyntaxError
    +-- SystemError
    +-- TypeError
    +-- ValueError
```

Multiple Exceptions

- Option 1:
 - Catch them in a **single** 'Except block' using them **as tuples**
- Option 2:
 - Catch them separately in **different** types of 'Except block'

Multiple Exceptions

- Option 1:
 - Catch them in a **single** 'Except block' using them as **tuples**

```
def main():  
    # multiple exceptions  
  
    try:  
        num = int(input("Enter a number: "))  
        key = 100//num  
        new_dict = {1:'a', 2:'b', 3:'c'}  
        print("Value at {} is {}".format( key, new_dict[key] ) )  
    except (ValueError, KeyError) as e:  
        print("Recovering from exception: {}, {} ".format(type(e), e))  
  
main()
```

Multiple Exceptions

```
def main():  
    # multiple exceptions  
  
    try:  
        num = int(input("Enter a number: "))  
        key = 100//num  
        new_dict = {1:'a', 2:'b', 3:'c'}  
        print("Value at {} is {}".format( key, new_dict[key] ) )  
    except (ValueError, KeyError) as e:  
        print("Recovering from exception: {}, {} ".format(type(e), e))  
  
main()
```

User enters number 50

```
Enter a number: 50  
Value at 2 is b
```

Multiple Exceptions

```
def main():  
    # multiple exceptions  
  
    try:  
        num = int(input("Enter a number: "))  
        key = 100//num  
        new_dict = {1:'a', 2:'b', 3:'c'}  
        print("Value at {} is {}".format( key, new_dict[key] ) )  
    except (ValueError, KeyError) as e:  
        print("Recovering from exception: {}, {} ".format(type(e), e))  
  
main()
```

User enters number 10

```
Enter a number: 10  
Recovering from exception: <class 'KeyError'>, 10
```


Multiple Exceptions

```
def main():  
    # multiple exceptions  
  
    try:  
        num = int(input("Enter a number: "))  
        key = 100//num  
        new_dict = {1:'a', 2:'b', 3:'c'}  
        print("Value at {} is {}".format( key, new_dict[key] ) )  
    except (ValueError, KeyError) as e:  
        print("Recovering from exception: {}, {} ".format(type(e), e))  
  
main()
```

User enters number 0

```
Enter a number: 0  
Traceback (most recent call last):  
  File "/Users/reza/Class_and_Research/drake_teaching/0  
    main()  
  File "/Users/reza/Class_and_Research/drake_teaching/0  
    key = 100//num  
ZeroDivisionError: integer division or modulo by zero
```

Multiple Exceptions

```
def main():  
    # multiple exceptions  
  
    try:  
        num = int(input("Enter a number: "))  
        key = 100//num  
        new_dict = {1:'a', 2:'b', 3:'c'}  
        print("Value at {} is {}".format( key, new_dict[key] ) )  
    except (ValueError, KeyError) as e:  
        print("Recovering from exception: {}, {} ".format(type(e), e))  
  
main()
```

User enters number 0

```
Enter a number: 0  
Traceback (most recent call last):  
  File "/Users/reza/Class_and_Research/drake_teaching/0  
    main()  
  File "/Users/reza/Class_and_Research/drake_teaching/0  
    key = 100//num  
ZeroDivisionError: integer division or modulo by zero
```

Added **ZeroDivisionError** in the Except Block

```
def main():  
    # multiple exceptions  
    try:  
        num = int(input("Enter a number: "))  
        key = 100//num  
        new_dict = {1:'a', 2:'b', 3:'c'}  
        print("Value at {} is {}".format( key, new_dict[key] ) )  
    except (ValueError, KeyError, ZeroDivisionError) as e:  
        print("Recovering from exception: {}, {}".format(type(e), e))  
  
main()
```

User enters number 0

```
Enter a number: 0  
Recovering from exception: <class 'ZeroDivisionError'>, integer division or modulo by zero
```

Multiple Exceptions

- Option 1:
 - Catch them in a **single** 'Except block' using them **as tuples**
- Option 2:
 - Catch them separately in **different** types of 'Except block'

Multiple Exceptions

```
def main_different_excepts():  
    # multiple exceptions  
    try:  
        num = int(input("Enter a number: "))  
        key = 100//num  
        new_dict = {1:'a', 2:'b', 3:'c'}  
        print("Value at {} is {}".format( key, new_dict[key] ) )  
  
    except ValueError as e:  
        print("Value issue : {}".format(e))  
  
    except KeyError as e:  
        print("Key issue : {}".format(e))  
  
    except ZeroDivisionError as e:  
        print("Divide by zero: {}".format(e))  
  
    except Exception as e:  
        print("Generic error: {}".format(e))  
  
main_different_excepts()
```

Multiple Exceptions

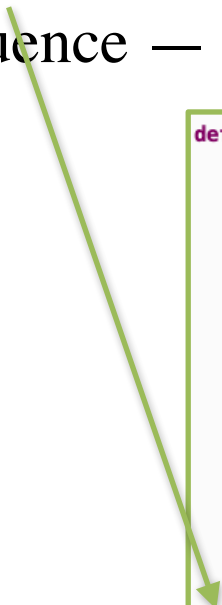
```
def main_different_excepts():  
    # multiple exceptions  
    try:  
        num = int(input("Enter a number: "))  
        key = 100//num  
        new_dict = {1:'a', 2:'b', 3:'c'}  
        print("Value at {} is {}".format( key, new_dict[key] ) )  
  
    except ValueError as e:  
        print("Value issue : {}".format(e))  
  
    except KeyError as e:  
        print("Key issue : {}".format(e))  
  
    except ZeroDivisionError as e:  
        print("Divide by zero: {}".format(e))  
  
    except Exception as e:  
        print("Generic error: {}".format(e))  
  
main_different_excepts()
```

Outputs for various inputs

```
>>> %Run exception2.py  
Enter a number: 0  
Divide by zero: integer division or modulo by zero  
  
>>> %Run exception2.py  
Enter a number: x  
Value issue : invalid literal for int() with base 10: 'x'  
  
>>> %Run exception2.py  
Enter a number: 10  
Key issue : 10
```

Multiple Exceptions

- The order of multiple except block is important:
 - act like a branch statement with multiple **elif** branches
 - a ‘catch all’ except block is usually placed at the end of the sequence — much like a **else** block



```
def main_different_excepts():  
    # multiple exceptions  
    try:  
        num = int(input("Enter a number: "))  
        key = 100//num  
        new_dict = {1:'a', 2:'b', 3:'c'}  
        print("Value at {} is {}".format( key, new_dict[key] ) )  
  
    except ValueError as e:  
        print("Value issue : {}".format(e))  
  
    except KeyError as e:  
        print("Key issue : {}".format(e))  
  
    except ZeroDivisionError as e:  
        print("Divide by zero: {}".format(e))  
  
    except:  
        print("Some unaccounted error occurred: {}".format(e))  
  
main_different_excepts()
```

Raise Exceptions

- Raise an exceptions to capture unwanted behavior by our program.

```
raise Exception (<<message>>)
```

- Message should be a string that describes the error
- Behaves the same as any other exception

Multiple Exceptions

- Raised exceptions can be handled the same way as any other python exception

```
def main_different_excepts():  
  
    # multiple exceptions  
    try:  
        num1 = int(input("Enter a positive number: "))  
        num2 = int(input("Enter another positive number: "))  
  
        if num1 <= 0 or num2 <= 0:  
            raise Exception("Entered number must be positive.")  
  
        print("Multiplication: ", num1*num2)  
        print("Division: ", num1/num2)  
  
    except (ValueError, KeyError, ZeroDivisionError) as e:  
        print("ValueError, KeyError, ZeroDivisionError: {}".format(e))  
  
    except Exception as e:  
        print("Newly raised error: {}".format(str(e)))  
  
main_different_excepts()
```

Multiple Exceptions

```
def main_different_excepts():  
    # multiple exceptions  
    try:  
        num1 = int(input("Enter a positive number: "))  
        num2 = int(input("Enter another positive number: "))  
  
        if num1 <= 0 or num2 <= 0:  
            raise Exception("Entered number must be positive.")  
  
        print("Multiplication: ", num1*num2)  
        print("Division: ", num1/num2)  
  
    except (ValueError, KeyError, ZeroDivisionError) as e:  
        print("ValueError, KeyError, ZeroDivisionError: {}".format(e))  
  
    except Exception as e:  
        print("Newly raised error: {}".format(str(e)))  
  
main_different_excepts()
```

Outputs for various inputs

```
Enter a positive number: 4  
Enter another positive number: 5  
Multiplication: {} 20  
Division: {} 0.8  
  
>>> %Run exception3.py  
  
Enter a positive number: -1  
Enter another positive number: 5  
Newly raised error: Entered number must be positive.
```