GOVERNMENT OF WESTERN AUSTRALIA

TAFE WA South Metropolitan

File I/O, Error Handling and Debugging

(Session 11)

Review (last week)

- Reading and writing Files
  - open() function returns a file object
  - Syntax: **open(filename, mode)**

```
file = open("myfile", "w")  # open file for writing
# write the data into the file
file.write("Hello World")
file.close()
file = open("myfile", "r")  # open file for reading
print(file.read())  # read file
```

```
Hello World
```

Overview

- Basic exception handling
- try – except block
- try – except – finally block
- Python assert Keyword
- Python Debugging Tools

Exceptions

- There are three types of errors:
  - **syntax errors** (mistakes in the source code, detected at compile-time)
  - **logical errors** – (produce incorrect results)
  - **exceptions** - (errors detected during execution)
- Sometimes the programs may *terminate or crash* unexpectedly
  - exceptions are error messages that indicate what happened
  - for instance, FileNotFoundError and ZeroDivisionError

Some Built-in Exceptions

- **BaseExeption –** is the base class of all exceptions. You can check the table showing built-in exception in Python.
- Some of the most common built-in exceptions in Python are:
  - **FileNotFoundError** – raised when a file or directory does not exists
  - **ZeroDivisionError** – raised when division by zero takes place
  - **ValueError –** raised when invalid values are specified
  - **NameError** – raised when an identifier is not found
  - **IndexError** – raised when an index is not found in a sequence

Python try-except Block

- Exceptions can be handled using *try–except* block of code
- The **try** block test your code for errors
- The **except** block is executed if an exception was raised in the try block
- Syntax:

```
try:
    # block of code
except:
    # block of code
```

Activity 1: FileNotFoundError

- It is raised when you try to execute a command that requires a file that the system cannot find

```
# open a text file for reading by using the open() function.
try:
    # open file for reading
    file = open("missing.txt", 'r')
    print(file.read())   # read file
except FileNotFoundError:     # if cannot find the path of “missing.txt”
    print("File not found")
```

Activity 2: ZeroDivisionError exception

- If an exception is raised, then except code is executed.

```
try:
    # Try to divide number by 0
    print(5 / 0) # result is infinite number
except:
    print("You can't divide by zero!")


try:
    # Handling divide by zero exception
    print(5 / 0)
except ZeroDivisionError:       # occurs when a number is divided by zero
    print("Caught ZeroDivisionError")
```

Activity 3: ValueError Exception

```
try:
    age = int(input("How old are you?: "))
    if age >= 18:
        print("You have a responsibility to vote in federal elections.")
    else:
        print("Not allowed to vote until you are 18 years of age.")
except ValueError:    # an invalid value for input
    print("Please enter a valid integer.")
```

```
====== RESTART: C:/Windows/System
How old are you?: 18
You have a responsibility to vote

============ RESTART: C:/Windows
How old are you?: we
Please enter a valid integer.
```

Activity 4: NameError

- It is raised when you try to use a variable that is invalid

```
my_number = 103
try:
    print(number)    # number is not defined
except NameError:     # when try to access a variable not defined
    print("Name error is raised.")
```

```
Name error is raised.
```

Activity 5: IndexError

- It is raised when your code try to access an index that is invalid (out of bounds)

```
 # Declaring and initializing list
list_fruits = ['apple', 'banana', 'kiwifruit', 'lime', 'orange']
try:
    # Print value of list at index 5
    print(list_fruits[5])
except IndexError:   # when an index is out of range
    print("Index is out of range")
```

```
Index is out of range
```

Multiple Exceptions

- There may be multiple except clauses. The following example uses two except blocks. If the “ValueError” exception is not occurred, then the except clause is executed.

```
x = '0'
try:    # check if an exception occurs inside the try clause
    y = int(input("Enter a number to be added: "))
    print(y + x)   # int + str?, type error
except ValueError:
    print(“Wrong value.")
except:
    print("Some error occurred.")
```

```
Enter a number to be added: w
Wrong value.

========== RESTART: C:/Wind
Enter a number to be added: 6
Some error occurred.
```

Finally Block

- *Finally* - it is an optional part of *try-except* statement
- It will be always executed
- Syntax:

```
try:
    # block of code
except:
    # block of code
finally:
    # block of code
```

Activity 6: Finally Block Example

```
balance = 2000.00
amount = input("Enter amount to be withdrawn: ")
try:
    if balance >= float(amount): # use {:.2f} to display 2 decimal places
        print("The current balance is ${:.2f}".format(balance-float(amount)))
    else:
        print("Insufficient funds")
except ValueError:
    print("Value Error occurred.")
except:
    print("Some Error occurred.")
finally:  # executed in any event
    print("Another transaction?")
```

```
Enter amount to be withdrawn: 4r
Value Error occurred.
Another transaction?
```

```
Enter amount to be withdrawn: 500
The current balance is $1500.00
Another transaction?
```

Assert Statement

- *Debugging* is the process of finding and removing errors (‘bugs’)
- The try-except block is one way of handling errors
- **assert** – is a Python debugging aid used to test a condition
- The condition should always be true, otherwise, if your code returns False, the program will raise an AssertionError
- Syntax:

assert expression, message

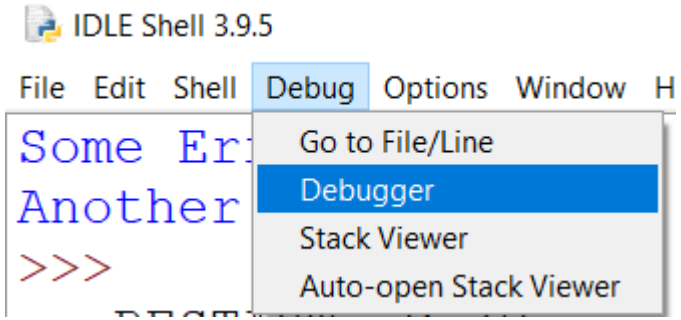
Activity 7: Assertions in Python Example

```
# the code inside the try block will be executed if here is no error,
# otherwise, it will be skipped, and the except block will be executed
try:
  num1 = 100
  num2 = 0
  assert num2 != 0 , "Division by zero, infinity does not exist!"
  print(num1 / num2)
except AssertionError as msg:    # AssertionError is thrown
  print(msg)  # error message given by user

print("An assert condition prevents the division by zero")
```

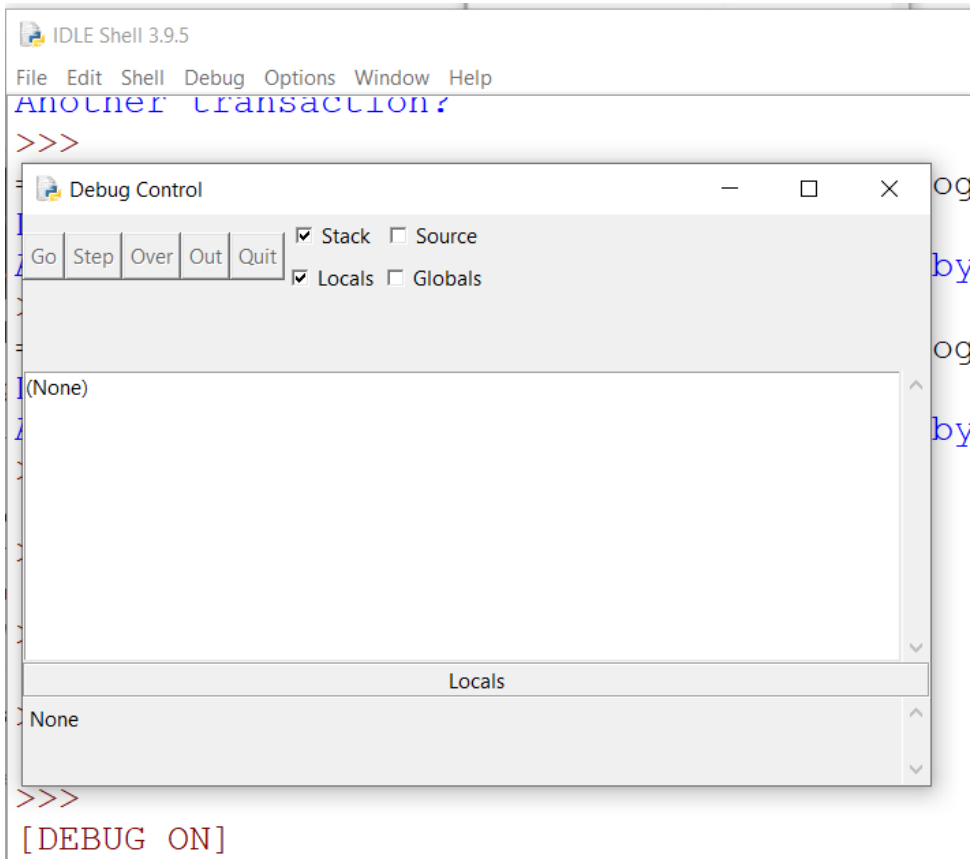
Python Debugging under IDLE

- Start the IDLE Shell and open the Debug window by selecting Debug => Debugger



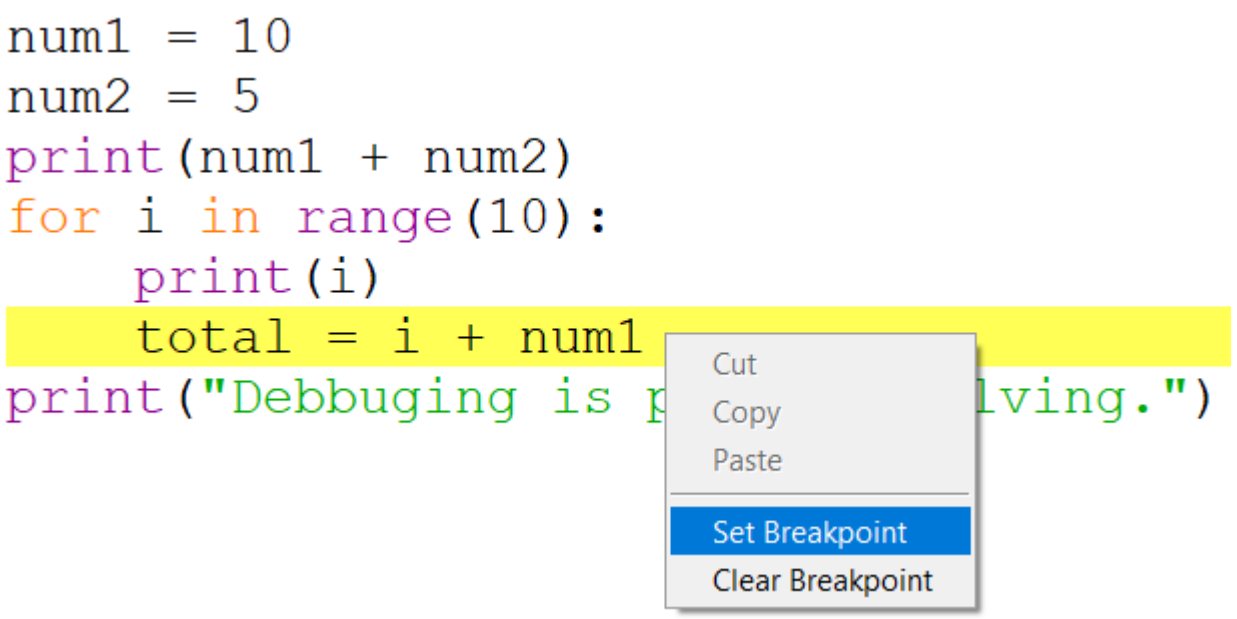
Debug Window (IDLE)

- Stepping Through Code
  - Go – tell the debugger to run your code
  - Step – step through one line at a time
  - Over – step over a function or loop
  - Quit – turn off the debugger
- Using Breakpoints
  - Instead of stepping though all lines, you can create a breakpoint where you want to investigate your code



IDLE Debugger Set or Clear Breakpoint (example)

- Right click on a line of your source and choose “set breakpoint”



Workshop 11

- Try-except statement
- How to Debug in IDLE
- Debugging with PyCharm

Questions?