Python

Error Handling

Program

```
def main():
    print('\n'*5)
    length = input('Enter length: ')
    height = input('Enter height: ')
    area = length * height
    print(' Area is -> ', area)
main()
```

err01

Program Errors

Generate the red line error messages

Example

```
Traceback (most recent call last):
  File "C:\DATA6\CCC-wright\CIS103\2021CIS103\Programs2021\python programs lecture\PGMS Error\err01.py", line 8, in <module>
    main()
  File "C:\DATA6\CCC-wright\CIS103\2021CIS103\Programs2021\python programs lecture\PGMS Error\err01.py", line 5, in main
    area = length * height
TypeError: can't multiply sequence by non-int of type 'str'
```

Program Errors

Generate the red line error messages

Example

```
Traceback (most recent call last):
  File "C:\DATA6\CCC-wright\CIS103\2021CIS103\Programs2021\python programs lecture\PGMS Error\err01.py", line 8, in <module>
    main()
  File "C:\DATA6\CCC-wright\CIS103\2021CIS103\Programs2021\python programs lecture\PGMS Error\err01.py", line 5, in main
    area = length * height
TypeError: can't multiply sequence by non-int of type 'str'
```

WHERE

BOTTON LINE IS THE PYTHON/SYSTEM MESSAGE

The program

```
def main():
    print('\n'*5)
    length = input('Enter length: ')
    height = input('Enter height: ')
    area = length * height
 area = length * height
TypeError: can't multiply sequence by non-int of type 'str'
    print(' Area is -> ', area)
main()
```

System generated errors

NOT SYNTAX

- Bad data (value error)
- Bad calculation(divide by zero)
- Undefined variable
- Input error
- User generated

Type checking

Form: type(var)

Check what kind of data type the variable is.

Try This

```
def main():
    a= input('enter a word: ')
    x =type(a)
    print(x)
    a = int(input('Enter a whole number: '))
    x = type(a)
    print(x)
    a = float(input('Enter a whole number: '))
    x = type(a)
    print(x)
main()
```

err02

Testing by type

By getting the data type of the variable

The data type can be tested.

Testing if variable is the right type

```
if type(a) is int:
```

Example:

```
t = type(a)
if t is int:
```

or

```
if type(a) is int
```

Also test for float & str

Testing data type

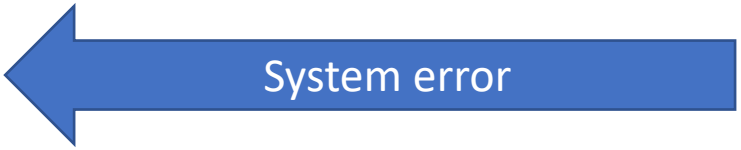
```
def testtype(a):
    if type(a) is int:
        print('Passed an integer')
    else:
        print(' -- not an integer --')
    return
def main():
    a = input('enter a word: ')
    testtype(a)
    print(type(a))
    print('-------')
    a = int(input('Enter a whole number: '))
    testtype(a)
    print(type(a))
    print('-------')
    a = float(input('Enter a decimal number: '))
    testtype(a)
    print(type(a))
main()
```

err03

Error Trapping

Syntax – basic form

```
try:
    statements
except:
    statements
```



Similar to if else try (if) except (else)

Add highlighted in yellow

NOTE

```
def main():
    length = input('Enter length: ')
    height = input('Enter height: ')
    try:
        area = float(length * float(height)
        print(' Area is -> ', area)
    except:
        print(' data error')
main()
```

GOOD

BAD

TRY THIS

```
def main():
    length = input('Enter length: ')
    height = input('Enter height: ')
    try:
        area=float(length) * float(height)
        print(' Area is -> ', area)
    except:
        print(' data error')
main()
```

err04

ANY Error conditions

except: <u>any error</u>

Specific Error condition

except exception:
specific error

Specific error handling with default

```
try:
    statements
except exception1:
    statements
except exception2:
    statements
except:
    statements
```

Specific exceptions

DEFAULT - required

Exception types

ValueError

NameError

ZeroDivisionError

TypeError

AttributeError

ValueError

- when a built-in operation or function receives an argument that has the right type but an inappropriate value, and the situation is not described by a more precise exception

NameError

- when a local or global name is not found. This applies only to unqualified names. The associated value is an error message that includes the name that could not be found.

ZeroDivisionError

- when the second argument of a division or modulo operation is zero. The associated value is a string indicating the type of the operands and the operation.

TypeError

- when an operation or function is applied to an object of inappropriate type. The associated value is a string giving details about the type mismatch.

AttributeError

- when an attribute reference or assignment fails.

```
def main():
    length = input('Enter length: ')
    height = input('Enter height: ')
    try:
        area = length * height
    except ValueError:
        print(' value error')
        area=float(length) * float(height)
    except TypeError:
        print(' type error')
        area=float(length) * float(height)
    except:
        area =0
    print(' Area is -> ', area)
main()
```

NOTE

SPECIFIC

DEFAULT

Try this

```
def main():
    length = input('Enter length: ')
    height = input('Enter height: ')
    try:
        area = length * height
    except ValueError:
        print(' value error')
        area=float(length) * float(height)
    except TypeError:
        print(' type error')
        area=float(length) * float(height)
    except:
        area =0
    print(' Area is -> ', area)
main()
```

err05

ALWAYS Set a default

```
try:
    statements
except exception1:
    statements
except exception2:
    statements
except:
    statements
```

Unspecified error
Always Last

Try this

```
def main():
    area = 0
    try:
        length = int(input('enter number: '))
        height = 2
        area = length * height
    except ValueError:
        print(' value error')
    except:
        print(' unknown error')
    print(' Area is -> ', area)
main()
```

err06

Multiple errors

```
try:
    statements
except (exception1, exception2) :
     statements
except:
      statements
```

```
def main():
    try:
        length = int(input('enter number: '))
        height = 2
        area = length * height
    except (ValueError, TypeError):
        print(' DATA ERROR')
    except:
        print(' unknown error')
    print(' Area is -> ', area)
main()
```

NOTE

Multiple Exception (specific)

DEFAULT

Try this

```
def main():
    try:
        length = int(input('enter number: '))
        height = 2
        area = length * height
    except (ValueError, TypeError):
        print(' DATA ERROR')
    except:
        print(' unknown error')
    print(' Area is -> ', area)
main()
```

err07

```
try:
    statements
except (exception1, exception2) :
    statements
except:
    statements
finally:
    statements
```

Clean up

Always executed

Try this

```
def main():
    area = 0
    try:
        length = int(input('enter number: '))
        height = 2
        area = length * height
    except (ValueError, TypeError):
        print(' DATA ERROR')
    except:
        print(' unknown error')
    finally:
        area=-1
    print(' Area is -> ', area)
main()
```

err08

Try … except error notes

try:

statements

execute so long as the data is good

except:

statements:

comes here if any statement under

the try fails

(red lines) message

Try – except

```
try:
    statement
    statement
    statement
    statement
except:
    statement
```

If any of these statements fail System message (red lines)

Comes here

Create your own error messages

Raising error

From a function, you can set and error value that will be picked up in the main line

Raise format

raise ValueError(message)

Which error to ‘raise’

Message as a string or a variable holding a string message

NOTE

```
def gethigh():
    tall=int(input('Enter height: '))
    if tall < 0:
        raise ValueError('invalid height can not be negative')
    return tall
```

Create the error
Note: error is NOT passed back by the return

NOTE

```
def main():
    again = "y"
    while again =='y':
        height = 0
        length = 0
        area = 0
        try:
            length = int(input('enter number: '))
            height = gethigh()
            area = length * height
        except ValueError as exerr:
            print(exerr)
        except:
            print(' unknown error')
        print(' Area is -> ', area)
        again = input('Again (y/n): ')
main()
```

While all is good

Try this part 1

```
def gethigh():
    tall=int(input('Enter height: '))
    if tall < 0:
        raise ValueError('invalid height can not be negative')
    return tall
```

Try this part 2

```
def main():
    again = "y"
    while again =='y':
        height = 0
        length = 0
        area = 0
        try:
            length = int(input('enter number: '))
            height = gethigh()
            area = length * height
        except ValueError as exerr:
            print(exerr)
        except:
            print(' unknown error')
        print(' Area is -> ', area)
        again = input('Again (y/n): ')
main()
```

TESTING

Testing

Your program should be able to handle anything a user does to input data.

Testing List

What happens if:

- Enter key is hit
- Spaces are entered
- Whole number entered
- Decimal number entered
- Alphabetic data in entered
- Special characters entered
- Mixed characters

Examples

Enter key is hit ----- (null character)

Spaces are entered -- blank (space key)

Whole number entered -- 23, -99, 10001

Decimal number entered -- 23.5, .009 , -10.10

Examples

Alphabetic data in entered -- AB ac xu

Special characters entered --- %& $ # @

Mixed characters -- A2 B$ 123# 82A

done