Exception Handling

- In the quadratic program, we used decision structures to avoid a run-time error
- Another mechanism called *exception handling* can also catch and deal with errors that arise while the program is running

Exception Handling

```
try:
    <body>
except <ErrorType>:
    <handler>
```

- Python `try`s to execute the statements inside the body
- If there is no error, control passes to the next statement after the `try-except`
- If an error occurs while executing the body, it looks for a matching error type. If one is found, the handler code is executed

Exception Handling

```
# quadratic.py
# Computes the real roots of a quadratic equation.
# Note: It crashes if no real roots.

import math
def main():
    print("Find the real solutions to a quadratic.")
    a, b, c = eval(input("Enter the coefs (a, b, c):"))
    discrim = b * b - 4 * a * c
    discRoot = math.sqrt(discrim)
    root1 = (-b + discRoot) / (2 * a)
    root2 = (-b - discRoot) / (2 * a)
    print("The solutions are:", root1, root2)

main()
```

Exception Handling

```
Finds the real solutions to a quadratic.
Enter the coefs (a, b, c): 1,2,3
Traceback (most recent call last):
  File "quadratic.py", line 14, in <module>
    main()
  File "quadratic.py", line 10, in main
    discRoot = math.sqrt(b * b - 4 * a * c)
ValueError: math domain error
```

Exception Handling

```
Finds the real solutions to a quadratic.
Enter the coefs (a, b, c): 1, 2
Traceback (most recent call last):
  File "quadratic.py", line 15, in <module>
    main()
  File "quadratic.py", line 8, in main
    a, b, c = eval(input("Enter the coefs (a, b, c):"))
ValueError: not enough values to unpack (expected 3, got 2)
```

Exception Handling

```
Finds the real solutions to a quadratic.
Enter the coefs (a, b, c): 1, 2, 3, 4
Traceback (most recent call last):
  File "quadratic.py", line 15, in <module>
    main()
  File "quadratic.py", line 8, in main
    a, b, c = eval(input("Enter the coefs (a, b, c):"))
ValueError: too many values to unpack (expected 3)
```

Exception Handling

```
Finds the real solutions to a quadratic.
Enter the coefs (a, b, c): 1, 2, n
Traceback (most recent call last):
  File "quadratic.py", line 15, in <module>
    main()
  File "quadratic.py", line 8, in main
    a, b, c = eval(input("Enter the coefs (a, b, c):"))
  File "<string>", line 1, in <module>
NameError: name 'n' is not defined
```

Exception Handling

```
Finds the real solutions to a quadratic.
Enter the coefs (a, b, c): 1, 2, '3'
Traceback (most recent call last):
  File "quadratic.py", line 15, in <module>
    main()
  File "quadratic.py", line 9, in main
    discrim = b * b - 4 * a * c
TypeError: unsupported operand type(s) for -: 'int' 
  and 'str'
```

Exception Handling

```
Finds the real solutions to a quadratic.
Enter the coefs (a, b, c): 1 2 3
Traceback (most recent call last):
  File "quadratic.py", line 15, in <module>
    main()
  File "quadratic.py", line 8, in main
    a, b, c = eval(input("Enter the coefs (a, b,
 c):"))
  File "<string>", line 1
    1 2 3
      ^
SyntaxError: invalid syntax
```

Exception Handling

```
# quadratic6.py
import math
def main():
    print("Find the real solutions to a quadratic.")
    try:
        a, b, c = eval(input("Enter the coefs (a, b, c):"))
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("The solutions are:", root1, root2 )
    except ValueError as excObj:
        if str(excObj) == "math domain error":
            print("VE: No Real Roots")
        else:
            print("VE: You didn't give me the right number of coefs.")
    except NameError:
        print("NE: A variable is not defined.")
```

Exception Handling

```
    except TypeError:
        print("TE: Your inputs were not all numbers.")
    except SyntaxError:
        print("SE: Inputs were not in the correct form.")
    except:
        print("Something went wrong, sorry!")
main()
```

Exception Handling

```
Find the real solutions to a quadratic.
Enter the coefs (a, b, c):1, 3, 2
The solutions are: -1.0 -2.0

Find the real solutions to a quadratic.
Enter the coefs (a, b, c):1, 2, 3
VE: No Real Roots

Find the real solutions to a quadratic.
Enter the coefs (a, b, c):1, 2
VE: You didn't give me the right number of coefs.

Find the real solutions to a quadratic.
Enter the coefs (a, b, c):1, 2, 3, 4
VE: You didn't give me the right number of coefs.
```

Exception Handling

```
Find the real solutions to a quadratic.
Enter the coefs (a, b, c):1, 2, n
NE: A variable name is not defined.

Find the real solutions to a quadratic.
Enter the coefs (a, b, c):1,2,'3'
TE: Your inputs were not all numbers.

Find the real solutions to a quadratic.
Enter the coefs (a, b, c):1 2 3
SE: Inputs were not in the correct form.
```

Exception Handling

- The multiple `try-except` act like `if-elif-else`
- The last bare `except` acts like an `else` and catches any errors without a specific match
- If there was no bare `except` at the end, the program could still crash

Max of Three

- Suppose we need an algorithm to find the largest of three numbers x1, x2, and x3

Strategy 1: Compare Each to All

```
if x1 >= x2 and x1 >= x3:
    max = x1
elif x2 >= x1 and x2 >= x3:
    max = x2
else:
    max = x3
```

Strategy 1: Compare Each to All

- What would happen if we were trying to find the max of five values?

Strategy 2: Decision Tree

```
if x1 >= x2:
   if x1 >= x3:
      max = x1
   else:
      max = x3
else:
   if x2 >= x3:
      max = x2
   else
      max = x3
```

Strategy 2: Decision Tree

- This approach is more complicated than the first

Strategy 3: Sequential Processing

- Initialize `max` with the first number in the list
- Scan through the list looking for a bigger number
- If you find a larger value, update `max`
- Continue looking until all the numbers in the list are checked

Strategy 3: Sequential Processing

```
# maxnum1.py
# Find the maximum of a series of numbers

def main():
    x1,x2,x3 = eval(input("Enter three values:"))
    max = x1
    if x2 > max:
        max = x2
    if x3 > max:
        max = x3
    print("The largest value is", max)

main()
```

Strategy 3: Sequential Processing

- Run the program

```
Enter three values:1,5,2
The largest value is 5
```

Strategy 4: Use Built-In Function

- Python has a built-in function called `max` that returns the largest of its parameters

```
# maxnum2.py
# Find the maximum of a series of numbers

def main():
    x1,x2,x3 = eval(input("Enter three values:"))
    print("The largest value is", max(x1, x2, x3))

main()
```

Strategy 3: Sequential Processing

- Run the program

```
Enter three values:1,5,2
The largest value is 5
```

Some Lessons

- There’s usually more than one way to solve a problem
- Don’t rush to code the first idea that pops out of your head
- Think about the design and ask if there’s a better way

Some Lessons

- Don’t reinvent the wheel
- If the problem you’re trying to solve is one that lots of other people have encountered, find out if there’s already a solution for it