Objectives

• Exception Handling

• Advanced Functions
What can go wrong when solving the quadratic equation?

This can be negative and produce an error.

How can we handle this problem?
We can use the decision structures, that is “if else” and check that the value is non-negative before using the math.sqrt function.
Example using the if-else statements

```python
import math

def main():
    print("This program finds the real solutions to a quadratic question")
    a = eval(input("Enter the co-efficient a"))
    b = eval(input("Enter the co-efficient b"))
    c = eval(input("Enter the co-efficient c"))
    discrim = b*b - 4*a*c
    if discrim < 0:
        print("The equation has no real solutions")
    elif discrim==0:
        root = -b / (2*a)
        print("The roots are equal and they are", root)
    else:
        discrim = math.sqrt(b*b - 4*a*c)
        root1 = (-b + discrim) / (2*a)
        root2 = (-b - discrim) / (2*a)
        print("The first root is", root1)
        print("The second root is", root2)
```
What are the other things that can go wrong in solving the quadratic equation?

\[ x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \]

And how would you handle that case?

The user might enter 0 for the value of a. Then we would be diving by 0 which is not meaningful.

Like previously, we can handle using the if-else statements.
The approach of using “if-else statements” looks like a good approach. But is there anything wrong with this approach?
The given quadratic question is a small program and involves just 2 or 3 checks for errors. But as you program, there maybe more than just 2 or 3 checks that you need to perform. And each time handling those cases itself might become longer than the actual code.

And in some cases you might not even know what checks that you need to perform.

So what is the better way to handle this?
Exception Handling is a better way of handling the runtime errors

You can write code to catch exceptions from a block of code.

The code reads like “Do the following series of steps, if an exception is raised, handle it this way.”

The program continues execution post the exception raising block
Exception Handling Syntax

A try block has the following form =>

Python will execute the code in <body> block

If an exception is raised, python looks for a matching “except” condition

try:
  <body>
except <ErrorType>:
  <handlerErrortype>
except:
  <handlerDefault>
else:
  <elseBlock>
finally:
  <finallyBlock>
Exception Handling Syntax

There can be multiple “except” blocks, there can be a default “except” block too.

If ErrorType match, or if it is default except block, python executes the corresponding handler.

try:
    <body>
except <ErrorType>:
    <handlerErrorType>
except:
    <handlerDefault>
else:
    <elseBlock>
finally:
    <finallyBlock>
Exception Handling Syntax

If no exception is raised, python will execute the else block.

In the end, python executes the finally block, which can be used to cleanup/undo any changes that were done in body.

```
try:
    <body>
except <ErrorType>:
    <handlerErrortype>
except:
    <handlerDefault>
else:
    <elseBlock>
finally:
    <finallyBlock>
```
Exception Handling Syntax

Python executes the finally block even if there was no exception.

The else and finally blocks are optional. However, there should be at least one except block.

```python
try:
    <body>
except <ErrorType>:
    <handlerErrortype>
except:
    <handlerDefault>
else:
    <elseBlock>
finally:
    <finallyBlock>
```
What is the output of the following code?

```python
try:
    a=1/0
except ZeroDivisionError:
    print("ZeroDiv")
except:
    print("DefaultExcept")
else:
    print("Else")
finally:
    print("Finally")
```
What is the output of the following code?

```python
try:
    a=1/0
except ZeroDivisionError:
    print("ZeroDiv")
except:
    print("DefaultExcept")
else:
    print("Else")
finally:
    print("Finally")
```

ZeroDiv
Finally
Using exception handling for the quadratic equation problem

```python
import math

def main():
    print("This program finds the real solutions to a quadratic equation")

    try:
        a = input("Please enter the co-efficient a")
        b = input("Please enter the co-efficient b")
        c = input("Please enter the co-efficient c")
        discriminant = math.sqrt(b*b - 4*a*c)
        root1 = (-b + discriminant) / (2*a)
        root2 = (-b - discriminant) / (2*a)
        print("The first root is", root1)
        print("The second root is", root2)
    except OverflowError:
        print("No real roots")
```

First try to find the roots and in case you encounter an error, run the except block
Some of the other types of Errors that can be caught using Exception Handling

**ValueError** : Error caused for using `math.sqrt()` with a negative number we have.

**ValueError**: User fails to type correct inputs (It is a different type of ValueError)

**NameError**: If user types an identifier instead of a number

**SyntaxError**: If input is not a valid Python expression ()

**TypeError**: If user types a valid Python expression that produces non-numerical results ()
Extending Exception Handling to handle different types of errors

```python
import math

def main():
    print("This program finds the real solutions to a quadratic equation")

    try:
        a = input("Please enter the co-efficient a")
        b = input("Please enter the co-efficient b")
        c = input("Please enter the co-efficient c")
        discrim = math.sqrt(b*b - 4*a*c)
        root1 = (-b + discrim) / (2*a)
        root2 = (-b - discrim) / (2*a)
        print("The first root is", root1)
        print("The second root is", root2)
    except OverflowError:
        print("No real roots")
    except ValueError:
        print("You did not give me three co-efficients")
    except NameError:
        print("You did not enter three numbers")
    except TypeError:
        print("Your inputs were not all numbers")
    except:
        print("Something went wrong, Sorry!")
```
Advanced Functions
What is the output of the following code?

```python
def my_avg(a, b):
    return (a+b)/2

def main():
    x = 5
    y = 10
    print(my_avg(x,y))
    avg = my_avg(3,4)
    print(avg)
    total = my_avg(4,3)*0.95
    print(total)

main()
```
What is the output of the following code?

def my_avg(a, b):
    return (a+b)/2

def main():
    x = 5
    y = 10
    print(my_avg(x, y)) # Function call takes place and the value returned is printed
    avg = my_avg(3, 4) # Function call takes place and then the returned value is used in assignment statement
    print(avg)
    total = my_avg(4, 3)*0.95 # Function call takes place and the returned value is used in an expression
    print(total)

main()
Functions can return multiple values

```python
def myfun(list_nos):
    total = 0
    prod = 1
    for i in range(len(list_nos)):
        total = total + list_nos[i]
        prod = prod * list_nos[i]
    return total, prod
```

```
x, y = myfun([1, 2, 3, 4])
print(x)
print(y)
```
Functions can return multiple values - Example Continued

```python
def myfun(list_nos):
    total = 0
    prod = 1
    for i in range(len(list_nos)):
        total = total + list_nos[i]
        prod = prod * list_nos[i]
    return total, prod
```

```python
x, y = myfun([1, 2, 3, 4])
print(x)
print(y)
```

The returned values are stored in the variables `x` and `y`

10
24
What is the output of the following?

```python
def getabc():
a = "Hello"
b = "World"
c = "!"
    return a, b, c
print(getabc())
```

```python
def func(x):
mul = x*2
div = x/2
    power = x**2
    return mul, div, power
```

```python
(a, b, c) = func(4)
print(a, b, c)
```
What is the output of the following?

```python
def func(x):
    mul = x*2
    div = x/2
    power = x**2
    return mul, div, power

a, b = func(4)
print(a, b)

def func(x):
    mul = x*2
    div = x/2
    power = x**2
    return mul, div, power

a, b, c, d = func(4)
print(a, b, c, d)
```

```python
>>> Traceback (most recent call last):
  File "C:/Users/Ajay/Desktop/Rec6/1.py", line 7, in <module>
    a, b = func(4)
ValueError: too many values to unpack (expected 2)

>>> Traceback (most recent call last):
  File "C:/Users/Ajay/Desktop/Rec6/1.py", line 7, in <module>
    a, b, c, d = func(4)
ValueError: need more than 3 values to unpack
```
# This program calculates the number of possible permutations of k objects from a set of n
# The Formula is given by: n! / (n-k)!
# Where n! = n*n-1*.....3*2*1

def factorial(n):
    prod = 1
    for i in range(1,n+1):
        prod = prod * i
    return prod

def permutation(n, k):
    n_fact = factorial(n)
    n_k_fact = factorial(n-k)
    return (n_fact/n_k_fact)

def main():
    n = 5
    k = 3
    perm = permutation(n, k)
    print("The permutation is", perm)
# This program calculates the number of possible permutations of k objects from a set of n
# The Formula is given by: n! / (n-k)!
# Where n! = n*n-1*.....3*2*1

def factorial(n):
    prod = 1
    for i in range(1, n+1):
        prod = prod * i
    return prod

def permutation(n, k):
    n_fact = factorial(n)
    n_k_fact = factorial(n-k)
    return (n_fact / n_k_fact)

def main():
    n = 5
    k = 3
    perm = permutation(n, k)
    print("The permutation is", perm)

main()
Functions can call another function-Continued

#This program calculates the number of possible permutations of k objects from a set of n
#The Formula is given by : n! / (n-k)!
#Where n! = n*n-1*.....3*2*1

```python
def factorial(n):
    prod = 1
    for i in range(1, n+1):
        prod = prod * i
    return prod

def permutation(n, k):
    n_fact = factorial(n)
    n_k_fact = factorial(n-k)
    return (n_fact/n_k_fact)

def main():
    n = 5
    k = 3
    perm = permutation(n, k)
    print("The permutation is", perm)
    main()
```

9) Call factorial function with argument (n-k). The function returns the result and it is stored in n_k_fact

10) The result is now returned to the main function

11) The result from the permutation function is now stored in the variable perm
# This program calculates the number of possible permutations of k objects from a set of n
# The Formula is given by: n! / (n-k)!
# Where n! = n*n-1*...3*2*1

def factorial(n):
    prod = 1
    for i in range(1, n+1):
        prod = prod * i
    return prod

def permutation(n, k):
    n_fact = factorial(n)
    n_k_fact = factorial(n-k)
    return (n_fact / n_k_fact)

def main():
    n = 5
    k = 3
    perm = permutation(n, k)
    print("The permutation is", perm)
main()
Functions need not have to “return” all the time

```python
import random

from graphics import *

def changeToRandomColor(Ball):
    r = random.randint(0, 255)
    g = random.randint(0, 255)
    b = random.randint(0, 255)
    color = color_rgb(r, g, b)
    Ball.setFill(color)

def main():
    win = GraphWin("MyWindow", 500, 500)
    win.setBackground("red")
    Ball = Circle(Point(250, 250), 20)
    Ball.draw(win)
    Ball.setFill("blue")
    while win.checkMouse()==None:
        changeToRandomColor(Ball)
        time.sleep(0.01)
    win.close()

main()
```

1) Execution starts here

2) Create a graphics window and set the Background to red

3) Create a circle and set the Fill to Blue
Functions need not have to “return” all the time

```python
import random

from graphics import *

def changeToRandomColor(Ball):
    r = random.randint(0, 255)
    g = random.randint(0, 255)
    b = random.randint(0, 255)
    color = color_rgb(r, g, b)
    Ball.setFill(color)

def main():
    win = GraphWin("MyWindow", 500, 500)
    win.setBackground("red")
    Ball = Circle(Point(250, 250), 20)
    Ball.draw(win)
    Ball.setFill("blue")
    while win.checkMouse() == None:
        changeToRandomColor(Ball)
        time.sleep(0.01)
    win.close()

main()  
```
Functions need not have to “return” all the time

```python
import random

from graphics import *

def changeToRandomColor(Ball):
    r = random.randint(0, 255)
    g = random.randint(0, 255)
    b = random.randint(0, 255)
    color = color_rgb(r, g, b)
    Ball.setFill(color)

def main():
    win = GraphWin("MyWindow", 500, 500)
    win.setBackground("red")
    Ball = Circle(Point(250, 250), 20)
    Ball.draw(win)
    Ball.setFill("blue")
    while win.checkMouse() == None:
        changeToRandomColor(Ball)
        time.sleep(0.01)
    win.close()

main()
```
Functions need not have to “return” all the time

```python
import random

from graphics import *

def changeToRandomColor(Ball):
    r = random.randint(0, 255)
    g = random.randint(0, 255)
    b = random.randint(0, 255)
    color = color_rgb(r, g, b)
    Ball.setFill(color)

def main():
    win = GraphWin("MyWindow", 500, 500)
    win.setBackground("red")
    Ball = Circle(Point(250, 250), 20)
    Ball.draw(win)
    Ball.setFill("blue")
    while win.checkMouse() == None:
        changeToRandomColor(Ball)
        time.sleep(0.01)
    win.close()

main()
```

7) `random.randint(a, b)` is a function defined in the random library. It returns a random integer N such that a<=N<=b. So here we generate a random number between 0 and 255
Functions need not have to “return” all the time

```python
import random

from graphics import *

def changeToRandomColor(Ball):
    r = random.randint(0, 255)
    g = random.randint(0, 255)
    b = random.randint(0, 255)
    color = color_rgb(r, g, b)
    Ball.setFill(color)

def main():
    win = GraphWin("MyWindow", 500, 500)
    win.setBackground("red")
    Ball = Circle(Point(250, 250), 20)
    Ball.draw(win)
    Ball.setFill("blue")
    while win.checkMouse() == None:
        changeToRandomColor(Ball)
        time.sleep(0.01)
    win.close()

main()
```

8) The function `color_rgb(red, green, blue)` will return a string representing a color that is a mixture of the intensities of red, green and blue specified. These should be ints in the range 0–255. Thus `color_rgb(255, 0, 0)` is a bright red, while `color_rgb(130, 0, 130)` is a medium magenta.
Functions need not have to “return” all the time

```python
import random

from graphics import *

def changeToRandomColor(Ball):
    r = random.randint(0, 255)
    g = random.randint(0, 255)
    b = random.randint(0, 255)
    color = color_rgb(r, g, b)
    Ball.setFill(color)

def main():
    win = GraphWin("MyWindow", 500, 500)
    win.setBackground("red")
    Ball = Circle(Point(250, 250), 20)
    Ball.draw(win)
    Ball.setFill("blue")
    while win.checkMouse() == None:
        changeToRandomColor(Ball)
        time.sleep(0.01)
    win.close()

main()
```

9) Set the color of the circle object. Note the function does not return anything. Now the control goes back to the main function.

Sleep is a function defined in the time library. It pauses for the amount specified. Here it pauses for 0.01 seconds.
Lists are mutable while int, float and string are non-mutable

Mutable Variables: Changes done to the variable in the called function will be reflected in the caller function

Non-Mutable: Changes done to the variable in the called function will NOT be reflected in the caller function
Passing Non-Mutable Variables: Example

```python
def test(x):
    x = x + 5

def main():
    x = 5
    test(x)
    print(x)

main()
```

Memory:

```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
```

5
Passing Mutable Variables: Example

```
def test(x):
    for i in range(len(x)):
        x[i] = x[i] + 10

def main():
    x = [10, 20, 30]
    test(x)
    print(x)

main()
```

```
[20, 30, 40]
```

Memory

<table>
<thead>
<tr>
<th>Memory</th>
<th>10</th>
<th>20</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[1]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[2]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
def replace_space_with_dot(str):
    newStr = ""
    for i in range(len(str)):
        if(str[i]==' '):
            newStr = newStr + "."
        else:
            newStr = newStr + str[i]
    return newStr

def main():
    str = "This has spaces"
    newStr = replace_space_with_dot(str)
    print("the old String is", str)
    print("the new replaces String is", newStr)

main()
Let us write function which takes a string as an argument and replaces the “ ” (space) character with “.” character

```python
def replace_space_with_dot(str):
    newStr = ""
    for i in range(len(str)):
        if(str[i]==" "):  
            newStr = newStr + "."
        else:
            newStr = newStr + str[i]
    return newStr

def main():
    str = "This has spaces"
    newStr = replace_space_with_dot(str)
    print("the old String is", str)
    print("the new replaces String is", newStr)

main()
```

Here we first initialize the newStr to be an empty String. Then we keep on appending it with the characters od str. If the character of str is “ ” (space) character, then we instead append it with the “.” character

the old String is This has spaces
the new replaces String is This.has.spaces
Thank You