

Scientific Programming in Python

Error Handling and Loops

Dr. Mohammed Hasanuzzaman
Room J102, Melbourne Building (Office)
e-mail: mohammed.hasanuzzaman@cit.ie
Website: <https://mohammedhasanuzzaman.github.io>



Error Handling

- Introduction to Repetition Structures
- The while Loop: a Condition-Controlled Loop
- The for Loop: a Count-Controlled Loop
- Calculating a Running Total and Sentinels
- Input Validation Loops
- Nested Loops

Quadratic equation example



```
import math

print("This program finds the real solutions to a  
quadratic\n")
a = float(input("Enter coefficient a: "))
b = float(input("Enter coefficient b: "))
c = float(input("Enter coefficient c: "))

discrim = b * b - 4 * a * c
if discrim < 0:
    print("\nThe equation has no real roots!")
elif discrim == 0:
    root = -b / (2 * a)
    print("\nThere is a double root at", root)
else:
    discRoot = math.sqrt(b * b - 4 * a * c)
    root1 = (-b + discRoot) / (2 * a)
    root2 = (-b - discRoot) / (2 * a)
    print("\nThe solutions are:", root1, root2)

print("Thank you for using my programme")
```

What if the user makes a mistake entering the data?



```
IPython console
Console 1/A

Enter coefficient a: three
Traceback (most recent call last):

  File "<ipython-input-1-e8385b2e5609>", line 1, in <module>
    runfile('/Users/diarmuid.grimes/Dropbox/COMP8042 - Analytical &
Scientific Programming/Lectures/3. Iteration in Python/
ErrorHandlingExamples.py', wdir='/Users/diarmuid.grimes/Dropbox/
COMP8042 - Analytical & Scientific Programming/Lectures/3. Iteration in
Python')

  File "/anaconda3/lib/python3.6/site-packages/spyder/utils/site/
sitecustomize.py", line 705, in runfile
    execfile(filename, namespace)

  File "/anaconda3/lib/python3.6/site-packages/spyder/utils/site/
sitecustomize.py", line 102, in execfile
    exec(compile(f.read(), filename, 'exec'), namespace)

  File "/Users/diarmuid.grimes/Dropbox/COMP8042 - Analytical &
Scientific Programming/Lectures/3. Iteration in Python/
ErrorHandlingExamples.py", line 11, in <module>
    a = float(input("Enter coefficient a: "))

ValueError: could not convert string to float: 'three'
```

IPython console History log

What if the user makes a mistake entering the data?



We would prefer the error to be handled and execute a graceful exit...

```
This program finds the real solutions to a quadratic  
  
Enter coefficient a: three  
Input error - please check your input values and run the program  
again  
Thank you for using my programme  
  
In [4]:
```

IPython console

History log

Exception Handling



The programmer can write code that catches and deals with errors that arise while the program is running, i.e., “Do these steps, and if any problem crops up, handle it this way.”

Exception Handling



- The `try` statement has the following form:

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

- When Python encounters a `try` statement, it attempts to execute the statements inside the body.
- If there is no error, control passes to the next statement after the `try...except`.

Exception Handling



- Instead of crashing, the exception handler prints a message indicating that there was a problem.
- The `try...except` can be used to catch *any* kind of error and provide for a graceful exit.

Quadratic equation example



```
import math
try:
    print("This program finds the real solutions to a quadratic\n")
    a = float(input("Enter coefficient a: "))
    b = float(input("Enter coefficient b: "))
    c = float(input("Enter coefficient c: "))

    discrim = b * b - 4 * a * c
    if discrim < 0:
        print("\nThe equation has no real roots!")
    elif discrim == 0:
        root = -b / (2 * a)
        print("\nThere is a double root at", root)
    else:
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\nThe solutions are:", root1, root2)
except:
    print("Input error - please check your input values and run the program again")

# this code will be executed if the input is valid or not
# but usually we just want to exit
print("Thank you for using my programme")
```

Temperature conversion example



try:

```
celsius = float(input("What is the Celsius temperature? "))
fahrenheit = 9 / 5 * celsius + 32
print("The temperature is", fahrenheit, "degrees fahrenheit.")
if fahrenheit >= 90:
    print("It's really hot out there, be careful!")
if fahrenheit <= 30:
    print("Brrrrrr. Be sure to dress warmly")
```

except:

```
print("Error - check your input and try again")
```

Validating input



The datatype works but does it make sense???

A program to convert a measurement in inches to cm

inchesToCM.py



```
inches = float(input("What is inches measurement? "))  
cm = inches * 2.54  
print(inches, "inches = ", cm , "cm")
```

A program to convert a measurement in inches to cm

inchesToCM.py



```
try:
    inches = float(input("What is inches measurement? "))
    cm = inches * 2.54
    print(inches, "inches = ", cm , "cm")
except:
    print("ERROR - please run the program again")
```

BUT...



- There remains a problem... can you spot it?
- HINT: think about test data that does not make sense...

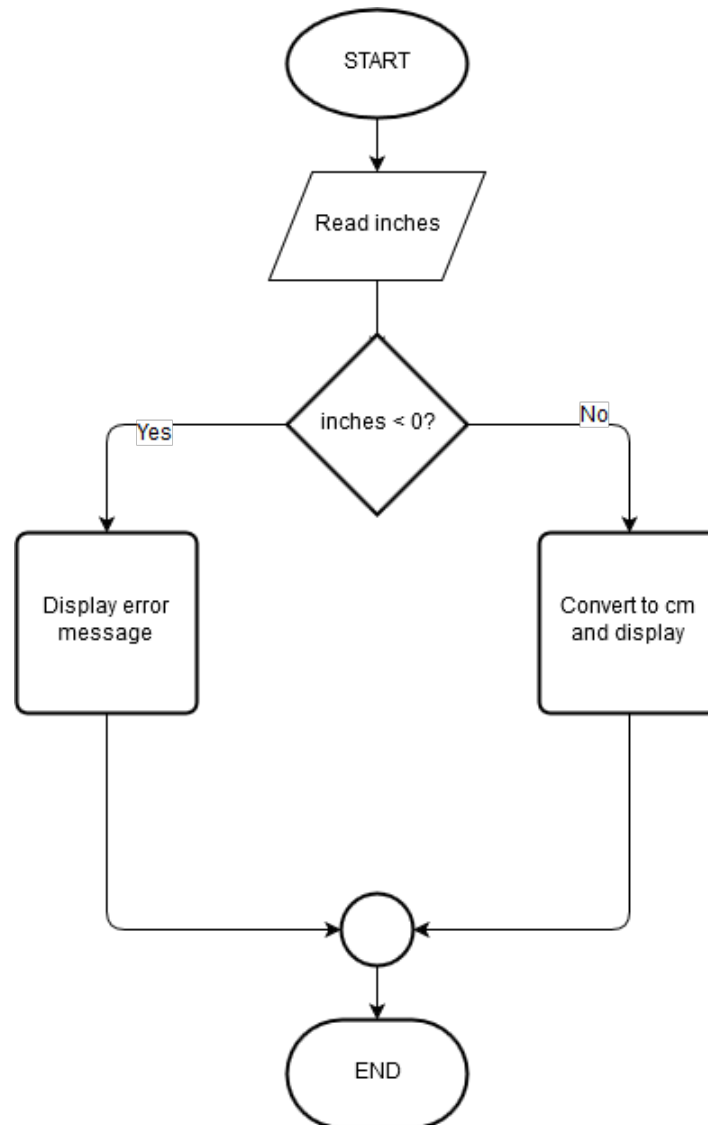
```
What is the inches measurement? -5.2  
-5.2 inches = -13.208 cm
```

BUT...



- The user is allowed to provide negative values.
- A negative length???
- It is a legal datatype but makes no sense!
- We can write an if statement to deal with this..

Here is the plan....



Note here i have
ignored data type

Coding the solution

inchesToCM2.py



```
try:
    inches = float(input("What is inches measurement? "))
    if inches < 0:
        print("ERROR - measurement cannot be negative")
    else:
        cm = inches * 2.54
        print(inches, "inches = ", cm , "cm")
except:
    print("ERROR - please run the program again")
```

Returning to our menu programme



```
menu = "Would you like " \
        "\n\t1: Coffee" \
        "\n\t2: Tea" \
        "\n\t3: Milk" \
        "\n==> "
```

- What user-errors can occur?
 1. User can type letters
 2. User can type a floating point number
 3. User can type a number greater than 3
 4. User can type a number less than 1
- (1) and (2) are handled by adding try/except
- (3) is handled using an if statement
- (4) is handled using an if statement
- (3) and (4) can be merged into an if/elif statement

What if not 1 or 2 or 3?

There are several ways to achieve this.. Here is one way



```
try:
    choice = int(input(menu))
    cost = 0
    if choice < 1:
        print("Invalid menu choice")
    elif choice > 3:
        print("Invalid menu choice")
    else:
        if choice == 1:
            cost = COFFEE
        elif choice == 2:
            cost = TEA
        elif choice == 3:
            cost = MILK
        print("That will be €" + str(cost))
except:
    print("Invalid input - please check your input")
```

```
COFFEE = 2.20
TEA = 1.70
MILK = 1.65
menu = "Would you like " \
       "\n\t1: Coffee" \
       "\n\t2: Tea" \
       "\n\t3: Milk" \
       "\n==> "
```

- Error Handling



Introduction to Repetition Structures

- The while Loop: a Condition-Controlled Loop
- The for Loop: a Count-Controlled Loop
- Calculating a Running Total and Sentinels
- Input Validation Loops
- Nested Loops

Introduction to Repetition Structures

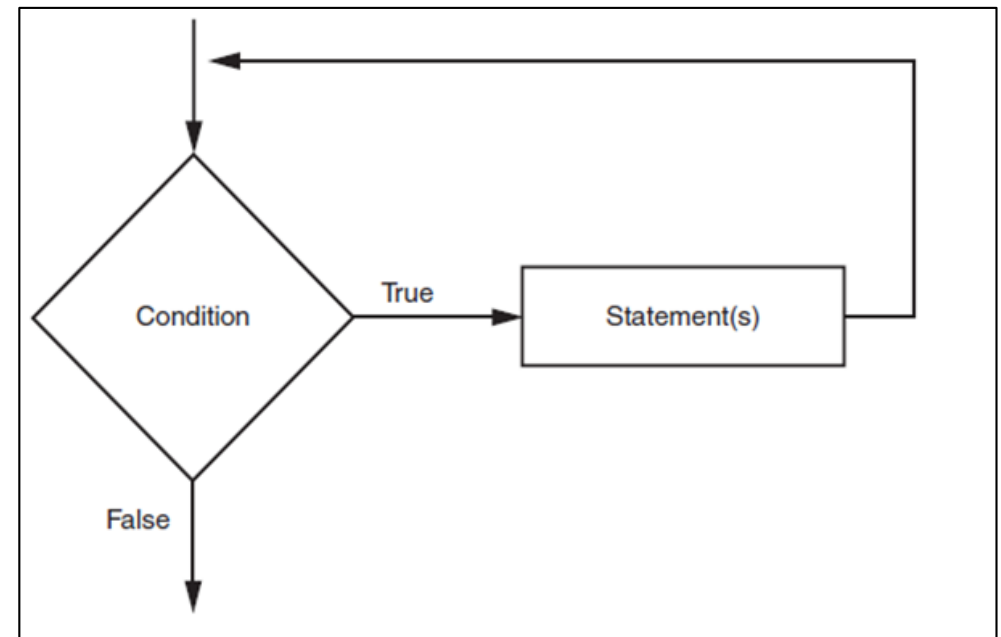


- We often have to write code that performs the same task multiple times
 - Disadvantages to duplicating code
 - Make programs large
 - Time consuming
 - May need to be corrected in many places
- All programming languages provide some kind of construct that support iterative actions

The `while` Loop: a Condition-Controlled Loop



- `while` loop: while condition is true, do something
 - Two parts:
 - **Condition** tested for true or false value
 - **Statements** repeated as long as condition is true
 - In flow chart, line goes back to previous part
 - General format:



```
while condition:  
    statements
```

The `while` Loop: a Condition-Controlled Loop



- Iteration: one execution of the body of a loop
- `while` loop is known as a *pretest* loop
 - Tests condition before performing an iteration
 - Will never execute if condition is false to start with
 - Requires performing some steps prior to the loop

```
num = 5
```

```
while num < 10:  
    print (num)  
    num = num+1
```

5
6
7
8
9

The `while` Loop: a Condition-Controlled Loop



```
num = 5

while num < 10:
    num = num+1
    print (num)
```

Typically last statement is increment of counter

The `while` Loop: a Condition-Controlled Loop



```
num = 5

while num < 10:
    print (num)
    num = num+1
```

- `num` is the *control variable*
- A control variable must change at some point during the loop otherwise the condition will not change

Example While Loop



```
# Create a variable to control the loop.
keepGoing = 'y'

while keepGoing == 'y':
    # Get a salesperson's sales and commission rate.
    sales = int(input ( 'Enter the amount of sales : ' ))
    commRate = int(input('Enter the commission rate: '))

    commission = sales * commRate
    print ('The commission is ' ,commission)

    keepGoing = input( 'Do you want to calculate another y/n')
```

```
stop = False
```

```
while stop == False:
```

```
    continueLooping = input( 'Do you want to continue looping y/n')
```

Infinite Loops



- Loops **must** contain within themselves a way to terminate
 - Something inside a `while` loop must eventually make the condition false
- Infinite loop: loop that does not have a way of stopping
 - Repeats until program is interrupted
 - Occurs when programmer forgets to include

Notice Boolean variable
stop is never True

```
stop = False
```

```
while stop == False:
```

```
    continueLooping = input( 'Do you want to continue looping y/n')
```

Infinite Loops

A screenshot of an IPython console window titled 'IPython console'. The window has a tab labeled 'Console 1/A'. The code being executed is a while loop that asks the user 'Do you want to continue looping y/n'. The loop is running, and the prompt is repeated multiple times. In the top right corner of the console window, there is a small red square button with a white 'X' icon, which is used to terminate the current command. A red arrow points from the text 'Terminates the current command' to this button. The bottom status bar shows 'End-of-lines: CRLF', 'Encoding: UTF-8', 'Line: 42', 'Column: 66', and 'Memory: 79 %'.

```
IPython 6.4.0 -- An enhanced Interactive Python.
In [1]: stop = False
...: while stop == False:
...:     continueLooping = input( 'Do you want to continue looping y/n')
...:
Do you want to continue looping y/nn
Do you want to continue looping y/nn
Do you want to continue looping y/nn
Do you want to continue looping y/nn
Do you want to continue looping y/nn
Do you want to continue looping y/nn
Do you want to continue looping y/nn
Do you want to continue looping y/nn
Do you want to continue looping y/nn
Do you want to continue looping y/nn
Do you want to continue looping y/n
```

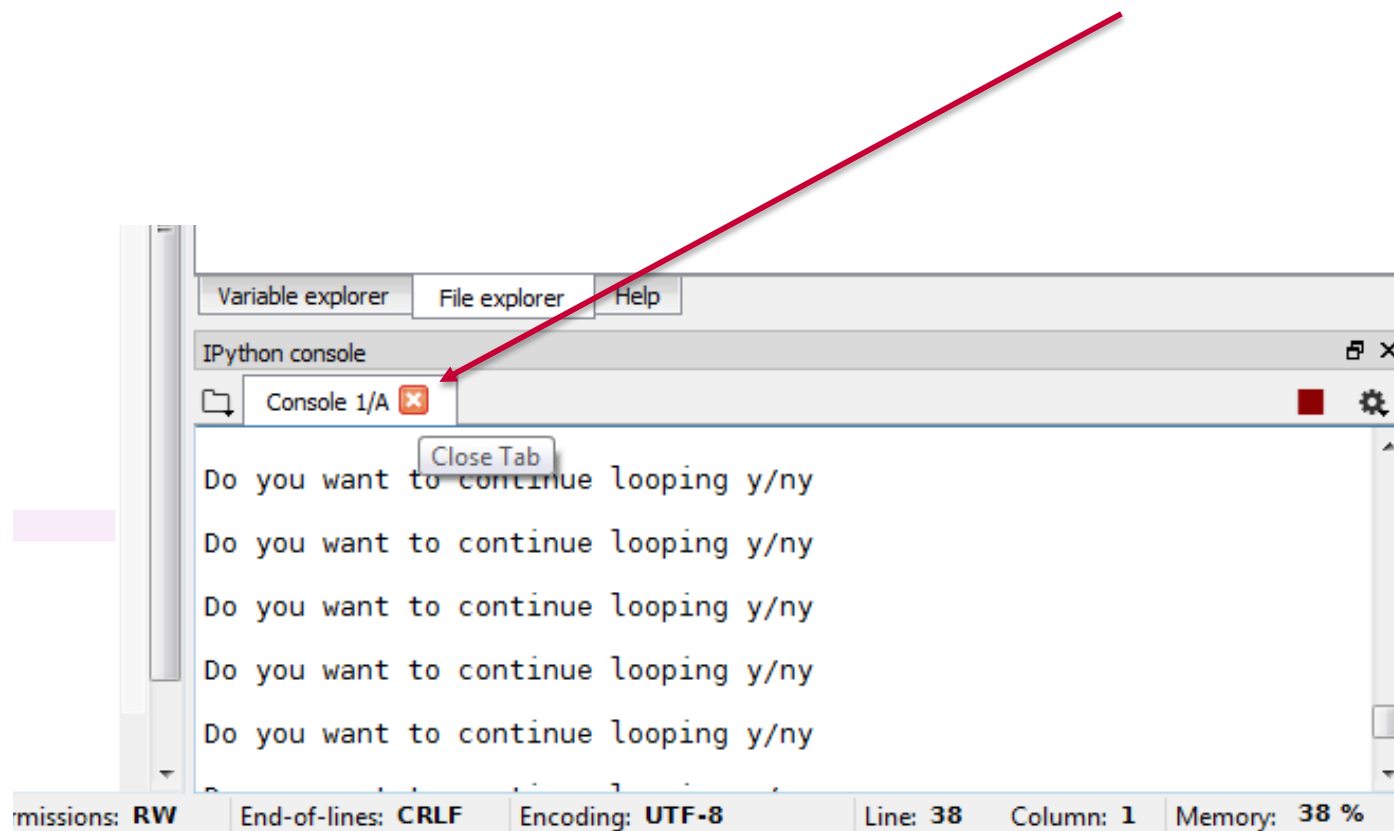
End-of-lines: CRLF Encoding: UTF-8 Line: 42 Column: 66 Memory: 79 %

Terminates the
current command

Terminating an infinite loop



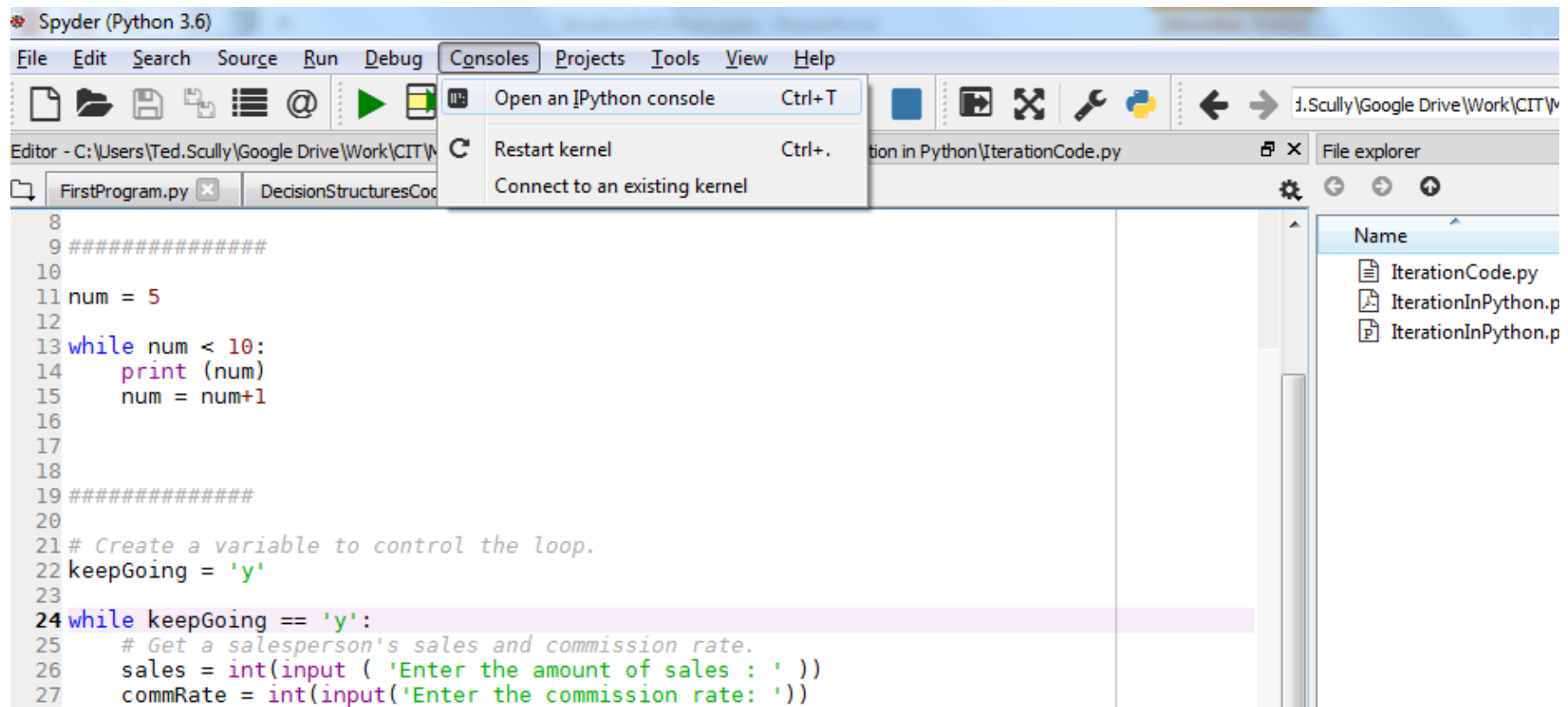
- Sometimes the approach demonstrated on the previous slide will not work and you will have to close and then restart the IPython console completely.
- To close the console just click the x on the console tab as shown below.



Terminating an infinite loop



- To restart the Ipython console you should select the Console dropdown menu and select “Open an IPython console” .
- Some versions of Spyder will automatically try to restart the console when you close it.



Example Infinite While Loop



```
num = 5
```

```
while num < 10:  
    print (num)
```


Is the example below an infinite While Loop



```
num1 = 5
num2 = 20

while num1<20 or num2>5:
    print (num1+num2)
    num1 = num1-1
    num2 = num2-1
```

Notice num1 must be less than 20 .
This is always true as num1 starts with a value of 5 and we keep decreasing it.

As this is an or logical operators once one of the operands is true the entire expression is true.

Exercise



- Write a program that will ask the user to continually guess a value between 1 and 10.
- When the user guesses the correct value the program should print out a message reporting they have been successful.

Exercise



```
number = 8
correctGuess = False

while correctGuess == False:

    userGuess = int(input( 'Please enter number'))

    if userGuess == number:
        print ("Well done. Correct")
        correctGuess = True
    else:
        print ("Plase try again")
```

The Augmented Assignment Operators



- In many assignment statements, the variable on the left side of the = operator also appears on the right side of the = operator
 - `num = num + 4`
- Augmented assignment operators: special set of operators designed for this type of job
 - Shorthand operators

Augmented assignment operators

Operator	Example Usage	Equivalent To
<code>+=</code>	<code>x += 5</code>	<code>x = x + 5</code>
<code>-=</code>	<code>y -= 2</code>	<code>y = y - 2</code>
<code>*=</code>	<code>z *= 10</code>	<code>z = z * 10</code>
<code>/=</code>	<code>a /= b</code>	<code>a = a / b</code>
<code>%=</code>	<code>c %= 3</code>	<code>c = c % 3</code>

Example



```
num = 5
```

```
while num < 10:  
    print (num)  
    num = num+1
```

```
num = 5
```

```
while num < 10:  
    print (num)  
    num += 1
```

Write an application that asks the user for a starting height of a bouncing ball.

At each bounce the ball halves it's height.

Display the height and number of each rebound bounce, stopping when the ball is 1mm off the ground.

```
This program computes the number and height of the rebounds of a dropped ball
```

```
What is the starting height?10
```

```
Rebound #1      :  5.000000m
```

```
Rebound #2      :  2.500000m
```

```
Rebound #3      :  1.250000m
```

```
Rebound #4      :  0.625000m
```

```
Rebound #5      :  0.312500m
```

```
Rebound #6      :  0.156250m
```

```
Rebound #7      :  0.078125m
```

```
Rebound #8      :  0.039062m
```

```
Rebound #9      :  0.019531m
```

```
Rebound #10     :  0.009766m
```

```
Rebound #11     :  0.004883m
```

```
Rebound #12     :  0.002441m
```

```
Rebound #13     :  0.001221m
```

```
Rebound #14     :  0.000610m
```

Things to note...



- It must stop when it is 1mm or less off the ground.
- $1\text{mm} = 0.001\text{m}$ so is our controlling value in the loop
i.e. while the height is greater than or equal to 0.001
keep iterating the loop

This program computes the number and height of the rebounds of a dropped ball

What is the starting height?10

Rebound # 1: 5.000000m

Rebound # 2: 2.500000m

Rebound # 3: 1.250000m

```
FINAL_BOUNCE = 0.001
```

```
print("This program computes the number and height "  
      "of the rebounds of a dropped ball")
```

```
bounceCount = 0
```

```
height = float(input("What is the starting height?"))
```

```
while height >= FINAL_BOUNCE:
```

```
    height /= 2
```

```
    bounceCount += 1
```

```
    print("Rebound #", format(bounceCount, "<4d"),  
          ":", format(height, "10.6f"), "m", sep=" ")
```


The `for` Loop: a Count-Controlled Loop



- Count-Controlled loop: iterates a specific number of times

- Use a `for` statement to write a count-controlled loop

- **Designed to work with sequence of data items**

- Iterates once for each item in the sequence

- General format:

```
for variable in [val1, val2, etc]:  
    statements
```

for Loop Example




```
print ('I will display the numbers 1 through 5.')  
for num in [1, 2, 3, 4, 5]:  
    print (num)
```

for Loop Example




The for loop

1st iteration:




```
for num in [1, 2, 3, 4, 5]:  
    print(num)
```

2nd iteration:




```
for num in [1, 2, 3, 4, 5]:  
    print(num)
```

3rd iteration:




```
for num in [1, 2, 3, 4, 5]:  
    print(num)
```

4th iteration:



```
for num in [1, 2, 3, 4, 5]:  
    print(num)
```

5th iteration:



```
for num in [1, 2, 3, 4, 5]:  
    print(num)
```

for Loop Example



```
for word in ["Hello", "There", "Everyone"]:  
    print (word)
```

Using the `range` Function with the `for` Loop



- The `range` function simplifies the process of writing a `for` loop
 - You can think off the range function as generating a list of numbers, which is generally used to iterate over with for loops.
 - In Python 3, when you use the range function it returns a range object, which the for loop can iterate over. Each time the for loop iterates the range object produces the next number in the list.

```
for num in range (5) :  
    print (num)
```

Notice that instead of using a list of values, we call to the range function passing 5 as an argument. Each time the for loop iterates the range function produces the next value in the list.

Using the range Function with the for Loop



- We can also force the range function to produce the entire list at once by using the list function.

```
# the range function generates a range object called  
numbers
```

```
numbers = range(5)  
print (type(numbers))
```

```
<class 'range'>  
[0, 1, 2, 3, 4]
```

```
# we can convert this to actual list by pass it as an  
argument to the list function  
print (list(numbers))
```

Using the `range` Function with the `for` Loop



```
range([start,] stop [,step])
```

- `range` characteristics:
 - **One argument:** used as **stopping** limit (not inclusive)
 - **Two arguments:** starting value and ending limit (not inclusive of the ending value)
 - **Three arguments:** third argument is step value

Using the range function with the for Loop



- `range` characteristics (all arguments **must be** integer):
 - One argument `range(stop)` : used as ending limit
 $0, 1, \dots, \text{stop}-1$
 - The default value for `start` is 0 and the default `step` is 1

Using the range Function with the for Loop



- In a for loop, the purpose of the target variable is to reference each item in a sequence of items as the loop iterates.
- In many situations it is helpful to use the target variable in a calculation or other task within the body of the loop.

```
for number in range(1, 9):  
    square = number**2  
    print (number, ' \t ' , square)
```

1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64

Using the range Function with the for Loop



- If you pass a third argument to the range function, that argument is used as a step value. Instead of increasing by 1, each successive number in the list will increase by the step value.

```
for num in range(1, 10, 2):  
    print (num)
```

1
3
5
7
9

Exercise



- Write the code to print all numbers divisible by 3 between 3 and 100.

```
for num in range(3, 100, 3):  
    print (num)
```

Letting the User Control the Loop Iterations



- We may have situations where we want the user to specify the number of iterations of a loop.
- Can **receive range inputs from the user**, place them in variables, and call the `range` function in the for clause using these variables
 - Be sure to consider the end cases: `range` does not include the ending limit
- Write a program that will ask the user to supply a *start* and *stop* number. Your program should then print out each number between *start* and *stop* raised to the power of 2.

User Controlled for Loop



```
print ('Program will print out squares of a sequence of values')
end = int(input('How high should I go? '))
start = int(input('Enter the starting number:'))

print ('Number \t Square')
print ('-----')

for number in range ( start, end + 1 ) :
    square = number**2
    print (number, ' \t ' , square)
```

```
How high should I go? 4
Enter the starting number:2
Number  Square
-----
2      4
3      9
4     16
```

Generating an Iterable Sequence that Ranges from Highest to Lowest



- The `range` function can be used to generate a sequence of numbers in descending order
 - How would I output the following sequence of numbers

10
9
8
7
6
5
4
3
2
1

```
for num in range (10, 0, -1):  
    print (num)
```

Make sure starting number is larger than end limit, and step value is negative

Input Validation Loops (cont'd.)



- Input validation: inspecting input before it is processed by the program
 - If input is invalid, prompt user to enter correct data
 - Basic validation commonly accomplished using a `while` loop which repeats as long as the input is bad
 - If input is bad, display error message and receive another set of data
 - If input is good, continue to process the input

Input Validation Example



```
# Get a test score.  
score = int(input('Enter a test score:'))  
  
# Make sure it is not less than 0.  
while score < 0 or score >100:  
    print ('ERROR: The score cannot be negative.\  
    score = int(input ('Enter the correct score:'))
```


Input Validation and Errors: previously we used the following pattern



try:

 read input

 process input

except:

 print error message

Example



```
try:
    number = int(input("Number >>> "))
    print("I will now continue with the rest of the program")
    print("Processing goes here...")
except:
    print("An error has occurred...exiting the programme")
```

- It works but....
- The messages do not help the user to identify the source of the problem
 - If the user has to enter 10 numbers, which number caused the problem?
 - It would be better to have an error message for each input
- Only gives the user one chance to enter data
 - It would be better to allow the user to re-try until the input is correct
- Solution?
 - Use try/except for each input
 - Use a while loop to loop until the user's input is ok

New Pattern for *all* numeric input



```
valid = False
while not valid:
    try:
        ask for input
        read input
        valid = True
    except:
        print error message
```

```
#assume false
# as long as the input is not valid....
# every language has their own try/except

# if the program has not crashed then the
user's input is valid
```

Example



```
ok = False
while not ok:
    try:
        number = float(input("Number >>> "))
        # if it reached here it must be ok
        ok = True
    except:
        print("Error with input")
```

Number >>> three

Error with input

Number >>> 55

Snippet of code to validate an exam mark



```
ok = False
while not ok:
    try:
        exam = float(input("Exam result >>> "))
        ok = (0 <= exam <= 100)
    except:
        print("Error with input - numbers only please")
```

Only true if user enters a number, and that number is between 0 and 100

Nested Loops



- Nested loop: loop that is contained inside another loop (remember nested if statements)

- Key points about nested loops:
 - Inner loop goes through all of its iterations for each iteration of outer loop
 - Example: **analog clock** works like a nested loop
 - Minutes hand moves 60 times for each movement of the hours hand: for each iteration of “hours” do 60 iterations of “seconds”
 - Total number of iterations in nested loop?

Nested For Loop Example



```
for num1 in range(3):  
    for num2 in range(2):  
        print (num1+num2)
```

0
1
1
2
2
3

Nested Loop Example



- Implement a program that will print out all possible times for the hours, minutes and seconds within a single day

```
for hours in range(24):  
    for minutes in range(60):  
        for seconds in range(60):  
            print (hours, ' : ' ,minutes, ' : ' , seconds)
```

Programming Task A



- Write a program that counts down in jumps of 10 from 200 to 0 (including 0)

200

190

180

170

....

Programming Task B



- Write a program that will allow the user to specify a start and stop numerical value.
- It should then display each number between the start and stop numerical values (should include both start and stop) and inform the user if it is an even or odd number

Solution Task A



```
for num in range(200, -1, -10):  
    print (num)
```

Solution Task B



```
start = int(input("Please enter first number"))
stop = int(input("Please enter second number"))

for num in range(start, stop+1, 1):
    if num%2 == 0:
        print (num, "is even")
    else:
        print (num, "is odd")
```