

Making Decisions

Wrestling with Python

Overview

- The Story so far....
- A Python program
- A word about comments
- Making Decisions
- Controlling the execution of statements
- Making a useful program

THE STORY SO FAR

Python in a nutshell

- Python lets us tell computers what to do
- We can give it single commands (immediate mode) or we can put together a sequence of commands that makes a program
- The computer uses a “Python interpreter” which works out the meaning of each statement and then performs it

Python programs

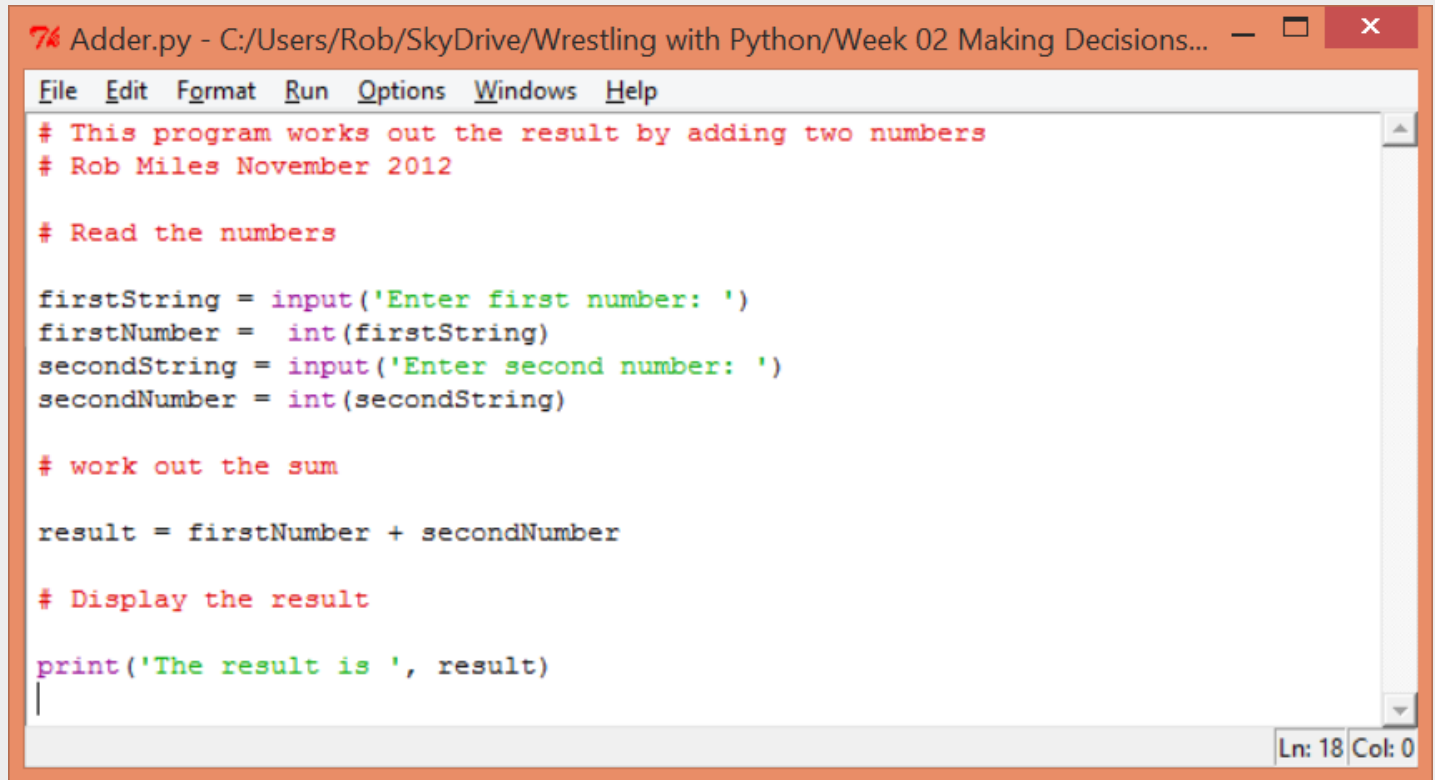
- A Python program is a sequence of statements which are obeyed in the order they are written
- Each statement can use a Python method (for example print) or it can work with data
- Data is stored boxes called “variables”
- Each variable has a name and a type

Creating Python

- We are using Python version 3.2.2 which provides a place we can write the programs and run them
- This is called IDLE
- We can store Python programs in text files that have the extension “.py” on the end of the filename

A PYTHON PROGRAM

Sums



```
74 Adder.py - C:/Users/Rob/SkyDrive/Wrestling with Python/Week 02 Making Decisions... - [ ] [X]
File Edit Format Run Options Windows Help
# This program works out the result by adding two numbers
# Rob Miles November 2012

# Read the numbers

firstString = input('Enter first number: ')
firstNumber = int(firstString)
secondString = input('Enter second number: ')
secondNumber = int(secondString)

# work out the sum

result = firstNumber + secondNumber

# Display the result

print('The result is ', result)
|
Ln: 18 Col: 0
```

- This program reads in two numbers, adds them together and then prints out the result

Sums

```
76 Adder.py - C:/Users/Rob/SkyDrive/Wrestling with Python/Week 02 Making Decisions... - [ ] [X]
File Edit Format Run Options Windows Help
# This program works out the result by adding two numbers
# Rob Miles November 2012

# Read the numbers

firstString = input('Enter first number: ')
firstNumber = int(firstString)
secondString = input('Enter second number: ')
secondNumber = int(secondString)

# work out the sum

result = firstNumber + secondNumber

# Display the result

print('The result is ', result)
|
Ln: 18 Col: 0
```

- Ask for the first number and read it in as a string

Sums

```

76 Adder.py - C:/Users/Rob/SkyDrive/Wrestling with Python/Week 02 Making Decisions...
File Edit Format Run Options Windows Help
# This program works out the result by adding two numbers
# Rob Miles November 2012

# Read the numbers

firstString = input('Enter first number: ')
firstNumber = int(firstString)
secondString = input('Enter second number: ')
secondNumber = int(secondString)

# work out the sum

result = firstNumber + secondNumber

# Display the result

print('The result is ', result)
|
Ln: 18 Col: 0

```

- Convert the string into an integer which we can do sums with

Sums

```
74 Adder.py - C:/Users/Rob/SkyDrive/Wrestling with Python/Week 02 Making Decisions... - [ ] [X]
File Edit Format Run Options Windows Help
# This program works out the result by adding two numbers
# Rob Miles November 2012

# Read the numbers

firstString = input('Enter first number: ')
firstNumber = int(firstString)
secondString = input('Enter second number: ')
secondNumber = int(secondString)

# work out the sum

result = firstNumber + secondNumber

# Display the result

print('The result is ', result)
|
```

Ln: 18 Col: 0

- Ask for the second number and convert that into a second integer

Sums

```
74 Adder.py - C:/Users/Rob/SkyDrive/Wrestling with Python/Week 02 Making Decisions... - [ ] [X]
File Edit Format Run Options Windows Help
# This program works out the result by adding two numbers
# Rob Miles November 2012

# Read the numbers

firstString = input('Enter first number: ')
firstNumber = int(firstString)
secondString = input('Enter second number: ')
secondNumber = int(secondString)

# work out the sum

result = firstNumber + secondNumber

# Display the result

print('The result is ', result)
|
Ln: 18 Col: 0
```

- Do the sum

Sums

```
74 Adder.py - C:/Users/Rob/SkyDrive/Wrestling with Python/Week 02 Making Decisions... - [ ] [X]
File Edit Format Run Options Windows Help
# This program works out the result by adding two numbers
# Rob Miles November 2012

# Read the numbers

firstString = input('Enter first number: ')
firstNumber = int(firstString)
secondString = input('Enter second number: ')
secondNumber = int(secondString)

# work out the sum

result = firstNumber + secondNumber

# Display the result

print('The result is ', result)
|
Ln: 18 Col: 0
```

- Display the result

A WORD ABOUT COMMENTS

Writing Software

- It is important when you write software that you ensure that you do it well
- A “good” program is not just one that works – although this does of course help
- For a program to be properly useful it is also important to ensure that it is well written

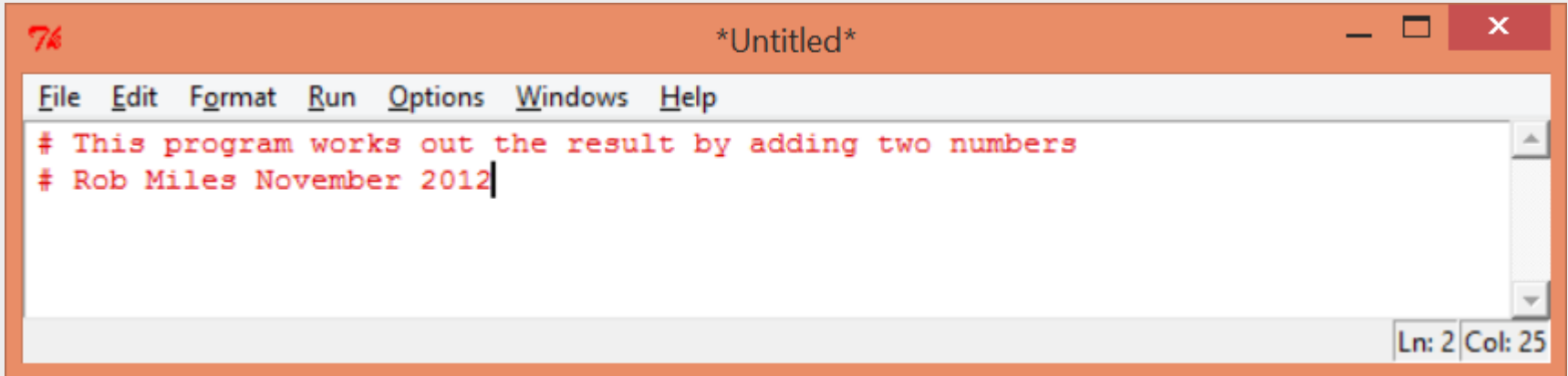
Well Written Code

- Easy to read
 - All the names in the text should add meaning
- Clean and consistent layout
 - The same format for common constructions
- Well managed
 - It should be clear who wrote the code and the reasons for any changes

Comments

- One way to add a lot of value to a program is to add comments
 - We already do this with sensible variable names, but comments allow even more detail
- A comment is something that the compiler completely ignores
 - It is only for use by the programmer

Creating a Comment



```
76 *Untitled*  
File Edit Format Run Options Windows Help  
# This program works out the result by adding two numbers  
# Rob Miles November 2012  
Ln: 2 Col: 25
```

- The character # means the start of a comment

Line Comments

```
x=0 # start at the left hand edge
```

- The character sequence # starts a comment that extends to the end of the line
- You can use these to quickly explain what a statement is doing

Stupid Comments

```
count = count + 1 # add 1 to count
```

- Comments should add value
- They should not just replicate information that a programmer should know already

Comments are cool

- Make sure that you use comments
- At least put your name and the date at the top
- That way you can convince yourself that you actually wrote the code when you look at it six months later....

MAKING DECISIONS

Program Flow

- At the moment every program we have written has just run through its statements in sequence
- This form of linear program flow is not always what you want
- The power of computer programs is that they can make decisions

The Three Types of Flow

1. Straight line:

Perform one statement after another

2. Decision:

Choose a statement based on a given condition

3. Loop

Repeat statements based on a given condition

Double Glazing Program

- We are going to consider a program we are writing for a customer
 - Read in height and width of window
 - Print out area and length of glass to buy
- This might even be useful
- Before we can write the program we need to go find some *metadata*

What is Metadata?

- Metadata is data about data
 - Limits (maximum and minimum values)
 - Units (measured in metres, gallons, years)
- It gives a proper context for what the program is doing
- You have to gather the Metadata **before** you write the program

Where does Metadata come from?

- It **must** come from the customer
 - They are the only people who can tell you about their business
- Only the double glazing salesman knows that he measure his windows in meters
- If you assume that he uses feet and inches you will supply a useless program

Getting Metadata

- You need to go out and ask the customer for this information
- They will not necessarily think to tell you
- Two assumptions that lead to disaster
 - Customer assumes you know the units
 - You assume the customer measures his windows in feet
- Result = **FAIL**

Double Glazing Metadata

```
# Window sizes measured in meters
# Invalid values:
#   width less than 0.5 metres
#   width greater than 5.0 metres
#   height less than 0.75 metres
#   height greater than 3.0 metres
```

- This is the metadata that drives our value inputs for the double glazing program
- I have written it as a comment
 - This is not accidental

Conditional Execution - if

- The if statement lets a program react in a particular way to data it receives
- This allows us to use metadata in our programs to make them more effective
 - The double glazing program could reject widths and heights that are incorrect
 - This will protect us from lawsuits..

Conditional Statement

```
if (condition):  
    statement we do if condition is true  
else:  
    statement we do if condition is false
```

- This is the general form of the Python conditional statement
- The condition is an expression that returns a boolean result – True or False

Relational Operators

```
2 * ( width + height ) * 3.25
```

- We have seen how a operators can be used in arithmetic expressions to produce numeric results

```
height > 3.0
```

- We can use relational operators in expressions to produce boolean results which are true or false

Testing the height upper limit

```
if (height > 3.0):  
    print('Too high')  
else:  
    print('OK')
```

- This test validates the upper bound of the height value
- Note that it doesn't check for heights which are too small or negative

Missing off the else part

```
if (height > 3.0):  
    print('Too high')
```

- If you don't need the else part you can leave it out
- Whether you have an else part depends on what you are trying to achieve with the code
 - Don't feel obliged to add one

Relational Operators

- You use relational operators to perform comparisons
- A relational operator works between two numeric operands
- It returns a boolean result which is either True or False

== operator

```
if ( age == 21 ):  
    print ( 'Happy 21st' )
```

- The == operator returns true if the two operands are equal
- Note that this is not the same as the = operator, which performs assignment

== operator and Floating Point

```
if ( average == 1.0f ):  
    print ( 'Average of 1' )
```

- Because floating point values can't be held exactly it is very dangerous to compare them for equality
- The condition may be unreliable because of errors in calculation

== operator and strings

```
if ( name == 'Rob' ):  
    print ( 'Hello Rob' )
```

- We can compare strings for equality
- The comparison is case sensitive
 - The string "rob" would not be recognised by the above code

The != operator

```
if ( name != 'Rob' ):  
    print ( 'You are not Rob' )
```

- The != (not equals) operator returns true if the operands are not equal to each other
- This can be used in the same way as the == operator

The < and > operators

```
if (width < 0.5 ):  
    print ('width too low')
```

- The < and > operators test for less-than and greater-than respectively
- Note that if the operands are equal the result is not true

The `<=` and `>=` operators

```
if (width >= 0.5 ):  
    print ('not too low')
```

- These work like `<` and `>`, but also include the case where the two are equal
- To invert a `<` you have to use a `>=`
- The code above inverts the previous test

The ! operator

```
if ( not False )  
    print ( 'not false is true' )
```

- The not operator can be used to invert a boolean value
- It works on one operand or expression in brackets

Combining Logical Operators

- Sometimes a program needs to combine a number of logical expressions
 - If the height is too wide or the height is too high
- Python provides operators that can be used in this way:
 - `and` for logical **and**
 - `or` for logical **or**

Testing both height limits

```
if (height > 3 or height < 0.5):  
    print('height invalid')  
else:  
    print('height OK')
```

- The Logical Operator **or** can be used to combine two conditions
- If one **or** other of the conditions is true the operator will return true

Inverting the Condition

```
if (height <= 3 and height >= 0.5):  
    print('height valid')
```

- This test inverts the condition to return true if the height is valid
- Note we have to invert the conditions **and** change the logical operator

Creating Blocks

```
if ( height > 5 ):  
    print('height restricted')  
    height = 5
```

- All the statements that are indented ‘underneath’ the if statement are controlled by that statement
- This is how Python does “blocks”

Creating Blocks

```
if ( height > 5 ):  
    print('height restricted')  
    height = 5  
print('This message is always printed')
```

- The print message is always obeyed because it is not indented like the other two

Indenting Pain

- Python is one of the few languages that uses this indenting technique to show how code is controlled by conditions
- Other languages use brackets to mark the start and the end
- **But in Python you must get your indenting right or code will fail**

What we are going to do...

- Get the IDLE development environment going
- Solve a problem by creating a program that reads in data and makes decisions based on the values supplied