CHAPTER

2

Working with Data

Stations Along the Way

- Recognizing different variable types
- Converting between variable types
- Creating a basic list
- List-slicing
- Adding and removing list items
- Using for loops to traverse lists
- Using ranges
- Testing code with the debugger
- Building a complete program

Programs Work with Different Types of Data

- ✓ **Text** is converted to a series of numbers, which are eventually converted to binary code.
- ✓ **Integers** are numbers without trailing decimal values. They're efficient but imprecise.
- ✓ **Floating point numbers** are numbers with decimal points.

Trouble with Numbers

- ✓ Examine `baddAdd.py`
- ✓ 3 + 7 = 37?
  - Python accepts input data as text.
  - The + (plus sign) *concatenates* (combines) text values.
  - It actually sees 3 *concat* 7.
  - So Python gladly combines the two strings, giving 37.

Operator Overloading

- ✓ The plus sign adds numbers.
- ✓ It's also used to combine strings.
  - The string manipulation is called *concatenation.*
  - For example, 'good ' + 'morning' becomes 'good morning'.
- ✓ Python concatenates strings but adds numbers.
- ✓ Sometimes it guesses wrong.

Converting Strings to Numbers

- ✓ You sometimes have to tell Python what type of data it's dealing with.
- ✓ The int() function is perfect for this:

```
>>> x = input("give me a number: ")
>>> y = int(x) + 10
>>> print y
```

- ✓ View `intAdd.py` for a partial fix to the `baddAdd.py` problem.

Floating-Point Values

- ✓ Computers can only approximate real numbers.
- ✓ The most common approximation is called a *float* (for floating-point real number).
- ✓ Python has a *double* (double-precision floating point).
- ✓ In Python, all floats are really doubles.

Doing Floating Math

- ✓ If the value has a decimal point, Python automatically makes it a float.
- ✓ If any value is a float, Python makes a float result:

```
>>> print (10 / 4.0)
2.5
>>> print (10.0 / 4)
 2.5
```

Using the float() Function

- ✓ You can also use the float() function to convert any value to a float:

```
>>> print float("4")
4.0
>>> print float(5)
5.0
```

- ✓ View `calc.py` for a complete example.

Storing Information in Lists

- ✓ Many programs will have large amounts of data.
- ✓ Data can be stored in lists.
- ✓ Python lists are similar to arrays in other languages.
  - Lists have interesting features in Python.

A List Example

- ✓ View `inventory.py`:

```
>>> inventory = [
    "toothbrush",
    "suit of armor",
    "latte espresso",
    "crochet hook",
    "bone saw",
    "towel"]
```

- ✓ A list is surrounded by square braces([]).
- ✓ Items are separated by commas (,).

Extracting Values from a List

- ✓ It's just like string slicing.
- ✓ Remember, indices come *between* elements.
- ✓ Also, index starts at zero:

```
>>> print (inventory[1:3])
['suit of armor', 'latte espresso']
>>> print (inventory[5])
towel
>>> (print inventory[-3])
crochet hook
```

Changing a List

- ✓ You can change the value of a specific element:

```
inventory[3] = "doily"
```

- ✓ You can append a new value to the end of a list:

```
inventory.append("kitchen sink")
```

- ✓ You can remove values from the list:

```
inventory.remove("kitchen sink")
```

- ✓ More list methods: >>> help("list")

Looping Through a List

- ✓ Often, you'll want to work with all the elements in your list.
- ✓ Python has a nice looping mechanism for this kind of thing.
- ✓ See `superHero.py`.

Introducing the for Loop

✓See `superHero.py`:

```
>>>heroes = [
    "Buffalo Man",
    "Geek Boy",
    "Wiffle-Ball Woman"
    ]

for hero in heroes:
 print ("Never fear,", hero, "is here.")
 print ("fin")
print ("vraiment la fin")


Never fear, Buffalo Man is here
Never fear, Geek Boy is here
Never fear, Wiffle-Ball Woman is here
```

How superHero Works

- ✓ Python creates a normal variable called hero.
- ✓ Non-array variables are sometimes called *scalars.*
- ✓ The code repeats once per element in the list.
- ✓ Each time through, hero has a new value.

The Python for Loop

- ✓ Requires a list or similar structure.
- ✓ Requires a scalar.
- ✓ Assigns each element to the scalar in turn.
- ✓ Similar to foreach in other languages.
- ✓ The for line ends in a colon (:).
  - Subsequent line(s) are indented.

Indentation and Python

- ✓ In many languages, indentation is purely a matter of style.
- ✓ Python uses indentation to determine how things are organized.
- ✓ You must indent all lines of a loop.
- ✓ Sloppy indentation won’t run.

Using the Debugger

- ✓ The debugger can show exactly what's going on.
- ✓ It's very useful when things go wrong.

1. In the main console window (not the text editor), choose debugger from debug menu.
2. Run program.

Controlling the Debug Console

- ✓ View `superHero.py` in debug mode.
- ✓ The Debug console shows current position in the program.

1. Use Step to move one step at a time.
2. Watch the progress.
3. Check variables in the Locals window.
4. Use the Quit button to finish the program.

Creating a Range

- ✓ Sometimes you want something to happen a certain number of times.
- ✓ You could make a list of numbers.
  - Technically it's a tuple, not a list, but that discussion can wait.
- ✓ That's what the range() function does:

```
>>> print range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- ✓ Create a range of 10 values from 0 to 9.

Range() Examples

- ✓ The range() function can take up to three arguments.
  - **range(a):** Make range from 0 to a-1
  - **range (a, b):** Make range from a to b-1
  - **range (a, b, c):** Make range from a to b-1, skipping c values each time

```
>>> print (range(2,5))
[2, 3, 4]
>>> print (range(5, 30, 5))
[5, 10, 15, 20, 25]
```

Using range() with a Loop

- ✓ The range() function makes it easy to create loops that happen any number of times (like a traditional for loop):

```
>>> for i in range(3):
        print ("now on lap %d" % i)

now on lap 0
now on lap 1
now on lap 2
```