The L Line

The Express Line to Learning

CHAPTER

3 Taking Control

Stations Along the Way

- Using the if statement
- Creating a condition
- Using else to work with false conditions
- Creating while loops
- Defining well-defined while loops
- Creating functions
- Managing variable scope
- Passing data into and out of functions
- Building a loop with multiple exits

Making Decisions with if

- ✓ Type the following code in the console:

```
>>> sky = "blue"
>>> if sky == "blue":
        print ("It's daytime!“)
```

- ✓ After you press enter twice, you'll see this result:

```
It's daytime!
```

Introducing Conditions

- ✓ A condition is an expression.
- ✓ It can be evaluated to true or false.
- ✓ It's often a comparison between a value and a variable.
- ✓ It can also be a Boolean variable.

The == Comparison

- ✓ In Python (and many other programming languages), == means comparison.
- ✓ Read `if x == y` as “if x is equal to y.”
- ✓ String comparisons are usually == (equal to) or != (not equal to).

The if Statement and Conditions

- ✓ The if statement uses a condition to branch the program's behavior.
- ✓ If the condition is true, the indented code will occur.
- ✓ The if statement must end with a colon(:).
- ✓ All subsequent indented code will occur only if the condition is true.
- ✓ See Guido.py.

Responding to a False Condition

- ✓See `GuidoOrNot.py`:

```
firstName = input("What is your first name? ")
print ("Nice to meet you, " + firstName + ".“)

if firstName == "Guido":
    print ("Hey, thanks for inventing Python!“)
else:
    print ("Have you seen Guido around?“)
```

How the else Clause Works

- ✓The else clause happens when the condition is False.
- ✓It ends with a colon.
- ✓All code indented after else happens if the condition isn’t true.

When to Use if - else

- ✓You have a condition.
- ✓You want some things to happen if the condition is true.
- ✓You want some other things to happen if the condition is false.
- ✓NOTE: else only works in the context of if. It never stands alone.

Using elif for Multiple Conditions

- ✓ Sometimes, you might want to check for several things.
- ✓ See `LinusOrGuido.py`:

```
if firstName == "Guido":
    print ("Thanks for writing Python“)
elif firstName == "Linus":
    print ("Linux Rocks!“)
else:
    print ("If you're going to be an open-source star,“)
    print ("you might need to get a cooler name.“)
```

Using the if - elif - else Structure

- ✓ Use an if statement to set up your first condition.
- ✓ Use one or more elif clauses to set up alternate conditions.
- ✓ Finish with an else to handle all unsupported cases.
  - Use a final else, even if you don't think you'll need it.

Numeric Comparison Operators

Symbol	Meaning
==	equal to
!=	not equal to
<	less than
>	greater than
<=	Less than or equal to
>=	Greater than or equal to

Using Numeric Comparisons with elif

- ✓See `raceAnnouncer.py`:
  - Loops through a ten-lap race
  - Announces various things at different points in the race
  - Uses an if - elif -else structure to manage commentary
  - Uses several numeric comparisons

Looping for a while

- ✓ The while loop is a flexible looping mechanism.
- ✓ It works with a condition (like the if statement).
- ✓ As long as the condition is true, the loop continues.
- ✓ As soon as the condition is evaluated false, the loop ends.

A Simple while Loop

- ✓See `minivan.py`:

```
tripFinished = "NO"
while tripFinished != "YES":
    tripFinished = raw_input("Are we there yet? ")
    tripFinished = tripFinished.upper()

print ("Can we go home now?“)
```

Creating a Sentry Variable

- ✓ Often, one variable controls access to the loop.
- ✓ This special variable is sometimes called the sentry variable.
- ✓ It acts like a sentry at a guard post, controlling access to a secret area.

Problems with while Loops

- ✓While loops are easy to build, but they can cause logic problems.
- ✓It's easy to make a loop that never happens.
- ✓You can also make loops that never end.
- ✓These problems aren't usually caught by the syntax checker.

What's Wrong with This Code?

- ✓ Here's a while loop that won't do what you expect:

```
i = 1
while i > 0:
    i = i + 1
    print (i)
```

More Problem Loops:

- ✓What will this code do?

```
i = 1
while i < 10:
    print ("Hi“)
```

- ✓Or this?

```
i = 1
while i < 10:
    j =  i + 1
    print ("Hi“)
```

Building a Well-Behaved while Loop

1. Create a sentry variable.
2. Initialize the sentry.
3. Create a condition including the sentry.
4. Write code inside the loop that ensures the condition can be triggered.

Introducing Functions

- ✓ As code gets more involved, you need an organizational scheme.
- ✓ Functions break programs into smaller, semi-independent segments.
- ✓ Like a song with a road map:
  - Verse 1
  - Chorus
  - Verse 2
  - Chorus

Writing a Program with Functions

- ✓ See `ants1.py`.
- ✓ Main code road map:

```
verse1()
chorus()
verse2()
chorus()
```

- ✓ All details are in the functions.
- ✓ The main program shows the big picture.

Features of a Function

- ✓ Begins with the `def` keyword
- ✓ Often has its own docstring
- ✓ Function name includes parentheses:
  - (empty for now)
- ✓ Colon (:) indicates the beginning of a block of code
- ✓ Subsequent lines are indented

Defining the chorus() Function

```
def chorus():
    """ prints chorus """
    print """
    ...and they all go marching
    down-
    to the ground-
    to get out-
    of the rain.
    Boom boom boom boom boom boom boom
    """
```

- ✓ The function simply prints out the chorus.
- ✓ Triple-quoted strings can be more than one line long.

Writing the Other Verses

- ✓Unindent to indicate the end of a function.
- ✓Write a new function for each verse.
- ✓Functions should be written before the road map (at least in these early programs).

Functions and Scope

- ✓A function is like a miniature program.
- ✓Variables created inside that function are destroyed when the function is finished.
- ✓This characteristic is called *variable scope,* and variables inside functions are called *local* variables.
- ✓Local variables prevent certain kinds of errors.

How Scope Works

- ✓See `scope.py`:

```
varOutside = "I was created outside the function"
print "outside the function, varOutside is: %s" % varOutside

def theFunction():
    varInside = "I was made inside the function"
    print ("inside the function, varOutside is: %s" % varOutside)
    print ("inside the function, varInside is: %s" % varInside)

theFunction()
print ("back outside the function, varOutside is: %s" %
varOutside)
# if I uncomment the next line, the program will crash
#print ("back outside the function, varInside is: %s" % varInside)
```

Explanation of scope.py

- ✓ varOutside was made outside the function. It's a *global* variable.
  - Global variables have values in all functions and the main program.
- ✓ varInside was made inside the function.
  - It has meaning only when the function is running.

Communicating with Functions

- ✓ It's a good thing that functions are closed off from the main program.
- ✓ Sometimes, you want to pass information in.
- ✓ Sometimes, you want to return data back.
- ✓ Python has support for this behavior.

Returning Data from a Function

- ✓See `ants2.py chorus()` function:

```
def chorus():
    output = """
    ...and they all go marching
    down-
    to the ground-
    to get out-
    of the rain.
    Boom boom boom boom boom boom boom
    """
    return output
```

- ✓Main program code changes:

```
print chorus()
```

Returning Data from chorus()

- ✓The chorus() function no longer prints directly to the screen.
- ✓Instead, it gathers data into a string variable output.
- ✓The final statement is return output.
- ✓The function passes the value of output to the code that called it.
- ✓print chorus() prints the returned value of the chorus() function.

Passing Values into a Function

- ✓ You can send data values into a function.
- ✓ The parentheses can contain one or more *arguments.*
- ✓ Arguments become *parameters* (essentially, local variables) inside the function.

The Parameterized verse()

```
def verse(verseNum):
    if verseNum == 1:
        distraction = "suck his thumb"
    elif verseNum == 2:
        distraction = "tie his shoe"
    else:
        distraction = "I have no idea"

    output = """
    The ants go marching %(verseNum)d by %(verseNum)d hurrah, hurrah!
    The ants go marching %(verseNum)d by %(verseNum)d hurrah, hurrah!
    The ants go marching %(verseNum)d by %(verseNum)d,
    The little one stops to %(distraction)s
    """ % vars()
    return output
```

Calling the Verse from the Main Program

- ✓ Now the verse requires an argument:

```
print (verse(1))
```

How the verse() Function Works

- ✓Verse is created with a parameter called verseNum.
- ✓When verse is called, it must include a numeric value.
- ✓That value is stored in verseNum.
- ✓The verse changes output based on the value of verseNum.

Dictionary Interpolation

- ✓ The improved verse() function shows another form of interpolation.
- ✓ The vars() function creates a *dict* (a set of name-value pairs) corresponding to all the variables the program knows.
- ✓ Using the variable name inside the interpolation placeholder puts a specific variable there.
- ✓ `print ("Hi there, %(user)s!" % vars())`

Creating a Main Loop

- ✓ Games often have one primary loop that repeats indefinitely.
- ✓ The main loop often exits in multiple ways:
  - The user chooses to quit.
  - There’s some loosing condition.
  - The user closes the window.

Building a Typical Main Loop

- ✓ See `password.py`.

1. Create a Boolean variable.
2. Use the variable as the condition.
3. Change the variable when the loop should exit.

password.py

```
keepGoing = True
correct = "Python"
tries = 3
while keepGoing:
    guess = raw_input("Please enter the password: ")
    tries = tries - 1

    if guess == correct:
        print ("You may proceed“)
        keepGoing = False
    else:
        print ("That's not correct.“)
        if tries <= 0:
            print ("Sorry. You only had three tries“)
            keepGoing = False
        else:
            print ("You have %d tries left" % tries)
```

Discussion Questions

- ✓Why is it important for computer programs to make decisions?
- ✓Why is it so important to prevent endless loops?
- ✓How does variable scope protect programmers?
- ✓Why might you need a loop with more than one exit condition?