

Controlling the Flow

Self-Review Questions

Self-review 3.1 The following expression is true when age is 21 and height is 120, or age is 21 and height is 140:

```
( age == 21 and height == 120 ) or ( age == 21 and height == 140 )
```

Can you write the expression with fewer terms?

Yes.

Self-review 3.2 Simplify the following Boolean expressions:

1. **a or (b and c) not a**
2. **b and c or False**
3. **a and b or c or (b and a)**
4. **a == True or b == False**

Self-review 3.3 'Nand' is an operator that is false only when both of its operands are true. Python doesn't have a nand operator, so how can we write an expression that is equivalent to $A \text{ nand } B$?

Self-review 3.4 Under what conditions would the following code sequences print 'B'?

1.

```
if thing1 > 10 :  
    print 'A'  
else:  
    print 'B'
```
2.

```
if thing1 > 10 :  
    print 'A'  
elif thing1 > 200 :  
    print 'B'
```
3.

```
if thing1 > 10 :  
    print 'A'  
if thing1 > 200 :  
    print 'B'
```

```

4.     if thing1 > 10 and thing1 < 10 :
        print 'A'
    else:
        print 'B'

```

Self-review 3.5 How many times will Python execute the code inside the following while loops? You should answer the question *without* using the interpreter! Justify your answers.

1.


```

i = 0
while i < 0 and i > 2 :
    print "still going ..."
    i = i+1

```
2.


```

i = 25
while i < 100 :
    print "still going ..."
    i = i - 1

```
3.


```

i = 1
while i < 10000 and i > 0 and 1 :
    print "still going ..."
    i = 2 * i

```
4.


```

i = 1
while i < 10000 and i > 0 and 0 :
    print "still going ..."
    i = 2 * i

```
5.


```

while i < 2048 and i > 0 :
    print "still going ..."
    i = 2 * i

```
6.


```

i = 1
while i < 2048 and i > 0 :
    print "still going ..."
    i = 2 * i

```
7.


```

for i in [] :
    print "foobar!"

```

Self-review 3.6 What extra assert statement would be worth adding to the set of assertions that test the contains function on page ?? and page ???

To ensure there is no problem if the string is the last letters.

```
assert contains ('hellohello', 'ohello') == True
```

To ensure there is no problem if the sub-string is a single letters.

```
assert contains ('hellohello', 'h') == True
```

Self-review 3.7 Explain the difference between the iterative version of factorial on page ?? and the following:

```

def factorial ( x ):
    if x < 0 :
        raise ValueError , 'Factorial not applicable to negative values.'
    elif x == 0 :
        return 1
    else :
        value = 1
        for i in range ( x , 1 , -1 ) :
            value *= i
        return value

```

The earlier one counts up, the above counts down.

Self-review 3.8 Compare the first power function (in Section ??) to the more efficient one in Section ?. How many multiplications does each implementation make to evaluate `power (3 , 5)`? Make sure you show your reasoning.

Self-review 3.9 Referring to the recursive function for generating numbers in the Fibonacci Sequence (page ??), how many calls to `fibonacci` are made to evaluate `fibonacci (5)`? Can you write down a general rule for how many calls must be made to evaluate `fibonacci (n)`?

Self-review 3.10 What lists do the following expressions evaluate to?

1. `range (1 , 100)`
2. `range (1 , 50 , 2)`
3. `range (1 , 25 , 3)`
4. `range (10 , -10 , -2)`

Self-review 3.11 The following code contains the names `a`, `b` and `c`. What is the scope of each of the them?

```

a = 10
for b in range ( 1 , 10 ) :
    c = b + 10
    print c

```

Self-review 3.12 What does the following code do? How might it be used in a larger program?

```

print 'Menu:'
print '1. Play game'
print '2. See high score table'
print '3. Exit'
i = -1
while i < 1 or i > 3:
    j = raw_input ( 'Enter choice: ' )
    try :
        i = int ( j )
    except :
        continue

```

Self-review 3.13 For the following recursive function, label the base cases and recursive cases. What do these functions do?

```
def m(a, b):
    def mm(a, b, c):
        if a == 0: return c
        else: return mm(a - 1, b, c + b)
    return mm(a, b, 0)

def d(a, b):
    def dd(a, b, c):
        if a < b: return c
        else: return dd(a - b, b, c + 1)
    return dd(a, b, 0)
```

Hint: You are almost certainly already familiar with these algorithms, just not in this form! You might want to try working out what these functions would return for some simple input values.

Self-review 3.14 We are to write some assert statements of the form:

```
assert (m(?, ?) == ?)
```

and

```
assert (d(?, ?) == ?)
```

to create tests for the two functions from Self-review 3.13. What assert statements would it be useful to put into a test? What parameter values might cause problems for `m` and `d`?

Self-review 3.15 What do the Python keywords **break** and **continue** do?

Self-review 3.16 What does the following print out?

```
for i in range(1, 10):
    for j in range(1, 10):
        print i * j,
    print
```

Self-review 3.17 What do the keywords **raise**, **try** and **except** mean?

Programming Exercises

Exercise 3.1 Implement a recursive Python function that returns the sum of the first n integers.

Exercise 3.2 Implement an iterative Python function that returns the sum of the first n integers.

Exercise 3.3 Implement an iterative Python function to generate numbers in the Fibonacci Sequence.

Exercise 3.4 While and for loops are equivalent: whatever you can do with one you can do with the other. In the following programs, convert the following while loops into for loops and the for loops into while loops. Of course, your new loops should give the same results as the old ones!

1.

```
for i in range ( 1 ,100 ) :
    if i % 3 == 2 :
        print i , "mod" , 3 , "= 2"
```
2.

```
for i in range(10):
    for j in range(i):
        print "*",
    print "
```
3.

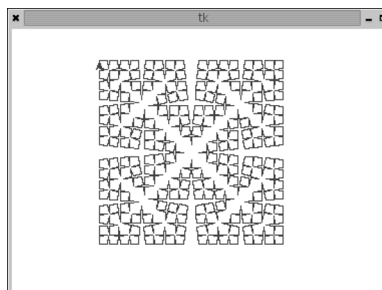
```
i = 0
while i < 100 :
    if i % 2 == 0 :
        print i , "is even"
    else :
        print i , "is odd"
    i = i + 1
```
4.

```
char = ""
print "Press Tab Enter to stop and Enter to keep going ..."
iteration = 0
while not char == "\t" and not iteration > 99:
    print "Keep going?"
    char = raw_input()
    iteration += 1
```

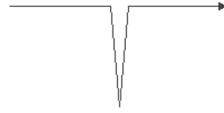
Exercise 3.5 Using books, the Web, or any other resource, find out what a logic gate is, and, in particular, what a half adder is. An output of the work should be a table showing the inputs and outputs for a half adder.

Implement a half adder as a few lines of Python code. Write a test program that shows that your implementation works as expected. You should be able to test that your code works for *all* possible input values.

Exercise 3.6 The following shows a Cesàro Fractal (also known as a Torn Square fractal). Write a program to draw this figure.



Hint: The Cesàro Fractal is based on a square. Each line, if it is longer than the minimum line length, is a line with an extra triangular section:



The triangular section is an isosceles triangle and is always on the 'inside' of the square.

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

# Even in Python 2.4.3, you still cannot have Unicode characters in function names :(

import turtle

def initialize ( color = 'blue' , smallest = 1.0 ) :
    """Prepare to draw Cesàro Fractal"""
    turtle.clear ()
    turtle.up ()
    turtle.goto ( -100.0 , 100.0 )
    turtle.setheading ( 0 )
    turtle.color ( color )
    turtle.down ()
    global smallestLineLength
    smallestLineLength = smallest

def cesaroLine ( overallLength ) :
    """Draw a line of the Cesàro Fractal of a given length. The global variable smallestLineLength
    assumed to be set."""
    halfSideLength = ( 0.9 * overallLength ) / 2.0
    if halfSideLength <= smallestLineLength :
        moveFunction = turtle.forward
    else :
        moveFunction = cesaroLine
        a = 85
        b = 180 - 2 * a
        moveFunction ( halfSideLength )
        turtle.right ( a )
        moveFunction ( halfSideLength )
        turtle.left ( 180 - b )
        moveFunction ( halfSideLength )
        turtle.right ( a )
        moveFunction ( halfSideLength )

def cesaroFractal ( overall , smallest = 1.0 , color = 'blue' ) :
    """Generate a Cesàro Fractal (a Torn Square Fractal) which is a square of lines with a 'triang
    initialize ( color , smallest )
    cesaroLine ( overall )
    turtle.right ( 90 )
    cesaroLine ( overall )
    turtle.right ( 90 )
    cesaroLine ( overall )
    turtle.right ( 90 )
```

```

cesaroLine ( overall )

if __name__ == '__main__':
    #initialize ( 'brown', 100 ) ; cesaroLine ( 200 )
    cesaroFractal ( 200.0 , 10.0 , 'brown' )
    raw_input ( 'Press any key to terminate' )

```

Exercise 3.7 Self-review 3.13 contained recursive implementations of functions `m` and `d`. Write equivalent functions that use iteration rather than recursion to produce the same results

Exercise 3.8 Write a function that capitalizes all the vowels in a string.

Hint: We have generally used indexing and the `range` function when using for loops. We can also use `for` loops for iterating over the characters in a string. For example:

```

for char in 'foobar':
    print char

```

Hint: Look at the documentation on strings: it has functions one or more of which are useful for this exercise. You might try using Python in interactive mode, importing the `string` module, and asking for help:

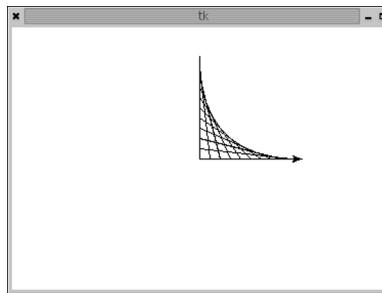
```

>>> import string
>>> help(string)
...

```

or look up the `string` module in the Python documentation.

Exercise 3.9 Use the Turtle module and nested loops to draw the following shape:



```

import turtle

turtle.goto (( 0 , 100 ))
turtle.goto (( 0 , 0 ))
turtle.goto (( 100 , 0 ))
turtle.goto (( 0 , 0 ))

for i in range ( 100 , -10 , -10 ) :
    turtle.up ()
    turtle.goto (( 0 , i ))
    turtle.down ()

```

```

turtle.goto (( 100 - i , 0 ))

raw_input ( 'Press Enter to close.' )

```

Exercise 3.10 Linux, Mac OS X, Solaris and other Unix-like systems have a command `cal` that, when run without a parameter, prints out the calendar for the current month. For example:

```

|> cal
          May 2007
Mo Tu We Th Fr Sa Su
   1  2  3  4  5  6
  7  8  9 10 11 12 13
 14 15 16 17 18 19 20
 21 22 23 24 25 26 27
 28 29 30 31

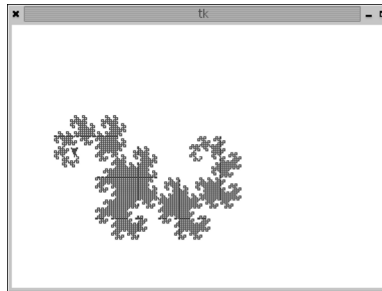
|>

```

Use loops and nested loops to print out a similar table for this month. Try to make sure that your columns line up correctly, like the example above!

Challenges

Challenge 3.1 The following shows a Dragon Curve.



Find an algorithm for drawing a Dragon Curve and write a program to draw a figure such as the one shown.

Hint: The starting point for a Dragon Curve is a straight line, but you always have to deal with two lines at a time. So this is not a simple recursion – more than one recursive function is needed.

```

#!/usr/bin/env python
# -*- coding: UTF-8 -*-

```

Even in Python 2.4.3, you still cannot have Unicode characters in function names :-)

```

import turtle

```

```

def initialize ( color = 'blue' , smallest = 1.0 ) :

```

```

    """Prepare to draw a Dragon Curve"""
    turtle.clear ()
    turtle.up ()
    turtle.setheading ( 0 )
    turtle.color ( color )
    turtle.down ()
    global smallestLineLength
    smallestLineLength = smallest

def dragonFirst ( overallLength ) :
    """Draw the first part of a pair of lines in the Dragon Curve."""
    if overallLength <= smallestLineLength :
        turtle.forward ( overallLength )
    else :
        overallLength /= 2.0
        dragonFirst ( overallLength )
        turtle.right ( 90 )
        dragonSecond ( overallLength )

def dragonSecond ( overallLength ) :
    """Draw the second part of a pair of lines in the Dragon Curve."""
    if overallLength <= smallestLineLength :
        turtle.forward ( overallLength )
    else :
        overallLength /= 2.0
        dragonFirst ( overallLength )
        turtle.left ( 90 )
        dragonSecond ( overallLength )

def dragonCurve ( overallLength , smallestLineLength , color ) :
    initialize ( color , smallestLineLength )
    dragonFirst ( overallLength )

if __name__ == '__main__' :
    #dragonCurve ( 2000.0 , 10.0 , 'purple' )
    #dragonCurve ( 4000.0 , 5.0 , 'purple' )
    dragonCurve ( 8000.0 , 2.5 , 'purple' )
    raw_input ( 'Press any key to terminate' )

```

Challenge 3.2 One application area in which iteration and recursion are particularly useful is user interfaces – programs that directly interact with people. These include websites, games, spreadsheets, and so on. For this challenge, you will create a text-based user interface to the Python turtle. The idea behind this is to let people use the turtle without having to learn Python first.

As a start, here is some code that allows you to move the turtle around using the keys a, s, w, and d on the keyboard:

```

import turtle

def turtle_interface () :
    """Allow the user to control the turtle via the keyboard."""
    steps = 10
    angle = 5
    while True :

```

```

i = raw_input ()
if i == 'w' : turtle.forward ( steps )
elif i == 's' : turtle.backward ( steps )
elif i == 'a' : turtle.left ( angle )
elif i == 'd' : turtle.right ( angle )
elif i == 'q' : break
else : continue

print 'Control the turtle!'
turtle_interface()

```

Improve this code by adding:

1. A function to print out a menu before the user starts. This should explain which keys are used to control the turtle. (Make sure that you keep the key map up to date as you add new functionality to the program.)
2. User options to perform the following functions:
 - (a) Clear the screen.
 - (b) Lift the 'pen' of the turtle off the page.
 - (c) Put the pen back down on the page.
 - (d) Change the width of the line that the turtle draws.
 - (e) Draw a circle. Better still, let the user decide how big a radius the circle should have.
 - (f) Change the color of the turtle. Allow the user to choose from a small number of colors (say, red, blue, green and black).

(You may well need to read through the documentation for the Turtle module to implement these.)

3. Undo capability. You will need to store the last key the user pressed and write some code (maybe in a separate function) to map keys onto their opposite actions. This might not be possible for some user options (such as drawing a circle) but others should be straightforward. For example, if the user has previously moved forward by 10 units, the opposite action is to move backwards 10 units.

Index

nand, 17

operator
nand, 17