_C6_90_ThinkPython-Chap7-Theory

April 2, 2020

1 7. Iteration

2 7.1 Multiple assignment

```
In [1]: bruce = 5
        print bruce,
        bruce = 7
        print bruce
```

```
5 7
```

```
In [2]: # Mathematical equality is a symmetric relation
        # if a=7 then 7=a
        #BUT the assignment is not symmetric.
        a=7
        7=a
```

```
          File "<ipython-input-2-ca16bc3a3039>", line 5
        7=a
    SyntaxError: can't assign to literal
```

```
In [8]: a=5
        b=a
        a=3
        # Are a and b equal?
        print a==b
        b=2
        print a
```

```
False
3
```

3 7.2 Updating variables

```
In [3]: x=x+1


        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)

        <ipython-input-3-174db80e4d10> in <module>()
    ----> 1 x=x+1


        NameError: name 'x' is not defined
```

```
In [7]: x = 1
        x = x+1 #increment
        print x
        x = x-1 #decrement
        print x
        x += 1.2
        print x
```

```
2
1
2.2
```

4 7.3 The while statement

```
In [8]: # recursion, chapter 5, page 44
        def countdown(n):
            if n<=0:
               print 'Out of space!'
            else:
               print n
               countdown(n-1)

        countdown(2)
```

```
2
1
Out of space!
```

```
In [9]: # repetition is iteration
        def countdown(n):
```

```
    while n>0:
        print n
        n = n - 1
    print 'Out of space!'

countdown(2)
```

```
2
1
Out of space!
```

```
In [ ]: """
        The flow of execution for a while statement:
            1. Evaluate the condition, yielding True or False
            2. If the condition is True
               then execute the body of the loop
                    including the condition updating
                    and then go back to step 1
               else (meaning the condition is False)
                    exit the while statement
                    and continue execution at the next statement after the while.
        """
```

```
In [16]: # LOOK OUT to the infinite loop - the condition never becomes True -
         # and the loop is repeated forever.
         # Does the next example involve an infinite loop?
         # Does this program terminate FOR ALL POSSIBLE VALUES of n?
         def sequence(n):
             while n!=1:
                 print n,
                 if n%2 == 0:
                     n = n/2
                 else:
                     n = n*3 + 1
             print n

         sequence(8)
         # Answer: NO ONE has been able to prove this program terminates
         # OR disprove it. See COLLATZ CONJECTURE
```

```
8 4 2 1
```

4.0.1 Exercise 7.1. (page 65)

Rewrite the function "print_n" from Section 5.8 using iteration instead of recursion.

```
In [ ]: def print_n(...):
            ...
```

5 7.4 break

```
In [22]: # You may use the break statement TO JUMP OUT of the loop.
         while True:
             line = raw_input('Your job is: ')
             if line == 'done':
                 break
             print line
         print 'Your job is done!'
         # It looks like a REPEAT - UNTIL repetition statement.
```

```
Your job is: d
d
Your job is: gg
gg
Your job is: ftf gg
ftf gg
Your job is: done
Your job is done
```

```
In [21]: # compare the previous code with the next one
         line = ''
         while line != 'done':
             line = raw_input('Your job is: ')
             print line
         print 'Your job is', line, '!'
```

```
Your job is: dd
dd
Your job is: sed fr
sed fr
Your job is: done
done
Your job is done
```

6 7.5 Square roots

```
In [ ]: # Newton's method for computing the square root of
        # a positive number a. If you start with a given estimate x
        # for the square root of a then the following code
        #evaluates y as a better estimate for the square root of a
        a = 4.0
        x = 3.0
        y = (x + a/x) / 2
        print y
```

```
In [ ]: x = y
        y = (x + a/x) / 2
        print y
        # execute this code many times!... :)
```

```
In [35]: # We can stop when x == y
         while True:
             print x
             y = (x + a/x) / 2
             if y == x: # !!!float equality
                 break
             x=y
```

```
2.00001024003
2.00000000003
2.0
```

```
In [ ]: # Rather than checking whether x and y are EXACTLY equal,
        # it is recommended to use a maths solution like this:
        #if abs(y-x) < epsilon:
        #    break
```

6.0.2 Exercise 7.2. (page 67)

Encapsulate this loop in a function called “square_root” that takes “a” as a parameter, choses a reasonable value of “x” and returns the estimate value of the square root of “a”.

```
In [ ]: def square_root(...):
            ...
```

7 7.6 Algorithms

```
In [ ]: """
        It is embarrassing that humans spend so much time IN SCHOOL
        learning to execute algorithms that,
        quite literally, require no intelligence.
        BUT, the process of DESIGNING ALGORITHMS is interesting,
        intelectually challenging and
        A CENTRAL PART of what we call PROGRAMMING.
        """
```

8 Debugging, Glossary, Exercises

8.0.3 Exercises 7.3. - 7.5. (page 69)