DALHOUSIE UNIVERSITY

Master of Engineering in Internetworking

Lab # 2

TurtleWorld, Conditions & Recursion, Fruitful Functions

INWK 6312

TurtleWorld

(Reference: Think Python: How to Think Like a Computer Scientist by Allen B. Downey)

A **package** is a collection of modules; one of the modules in Swampy is `TurtleWorld`, which provides a set of functions for drawing lines by steering turtles around the screen.

If Swampy is installed as a package on your system, you can import `TurtleWorld` like this:

```
from swampy.TurtleWorld import *
```

If you downloaded the Swampy modules but did not install them as a package, you can either work in the directory that contains the Swampy files, or add that directory to Python's search path. Then you can import `TurtleWorld` like this:

```
from TurtleWorld import *
```

Create a file named `test.py` and type in the following code:

```
from swampy.TurtleWorld import *

world = TurtleWorld()
bob = Turtle()
print bob

wait_for_user()
```

The first line imports everything from the `TurtleWorld` module in the `swampy` package.

The next lines create a TurtleWorld assigned to `world` and a Turtle assigned to `bob`. Printing `bob` yields something like:

```
<TurtleWorld.Turtle instance at 0xb7bfbf4c>
```

This means that `bob` refers to an **instance** of a Turtle as defined in module `TurtleWorld`. In this context, "instance" means a member of a set; this Turtle is one of the set of possible Turtles.

`wait_for_user` tells TurtleWorld to wait for the user to do something.

TurtleWorld provides several turtle-steering functions: `fd` and `bk` for forward and backward, and `lt` and `rt` for left and right turns. Also, each Turtle is holding a pen, which is either down or up; if the pen is down, the Turtle leaves a trail when it moves. The functions `pu` and `pd` stand for "pen up" and "pen down.

Question 1A

Write a function called `square` that takes a parameter named `t`, which is a turtle. It should use the turtle to draw a square.

Question 1B

Write a function call that passes `bob` as an argument to `square`, and then run the program again.

Question 2

Add another parameter, named `length`, to `square`. Modify the body so length of the sides is `length`, and then modify the function call to provide a second argument. Run the program again. Test your program with a range of values for `length`.

Question 2B

The functions `lt` and `rt` make 90-degree turns by default, but you can provide a second argument that specifies the number of degrees. For example,

`lt(bob, 45)` turns `bob` 45 degrees to the left.

Make a copy of `square` and change the name to `polygon`. Add another parameter named `n` and modify the body so it draws an n-sided regular polygon. Hint: The exterior angles of an n-sided regular polygon are 360/*n* degrees.

Question 3

Write a function called `circle` that takes a turtle, `t`, and radius, `r`, as parameters and that draws an approximate circle by invoking `polygon` with an appropriate length and number of sides. Test your function with a range of values of `r`.

Hint: figure out the circumference of the circle and make sure that `length * n = circumference`.

Another hint: if `bob` is too slow for you, you can speed him up by changing `bob.delay`, which is the time between moves, in seconds.

`bob.delay = 0.01` *ought to get him moving.*

Question 4

Make a more general version of `circle` called `arc` that takes an additional parameter `angle`, which determines what fraction of a circle to draw. `angle` is in units of degrees, so when `angle=360`, `arc` should draw a complete circle.

SECTION B

Reference: Think Python: How to Think Like a Computer Scientist by Allen B. Downey

modulus operator: An operator, denoted with a percent sign (`%`), that works on integers and yields the remainder when one number is divided by another.

boolean expression: An expression whose value is either `True` or `False`.

relational operator: One of the operators that compares its operands: `==`, `!=`, `>`, `<`, `>=`, and `<=`.

logical operator: One of the operators that combines boolean expressions: `and`, `or`, and `not`.

conditional statement: A statement that controls the flow of execution depending on some condition.

condition: The boolean expression in a conditional statement that determines which branch is executed.

compound statement: A statement that consists of a header and a body. The header ends with a colon (:). The body is indented relative to the header.

branch: One of the alternative sequences of statements in a conditional statement.

chained conditional: A conditional statement with a series of alternative branches.

nested conditional: A conditional statement that appears in one of the branches of another conditional statement.

recursion: The process of calling the function that is currently executing.

base case: A conditional branch in a recursive function that does not make a recursive call.

infinite recursion: A recursion that doesn't have a base case, or never reaches it. Eventually, an infinite recursion causes a runtime error.

Question 1A

Write a function that checks if a number is even or not.

Question 1B

Write a function called `do_n` *that takes a function object and a number,* `n`*, as arguments, and that calls the given function* `n` *times.*

Question 2

Fermat's Last Theorem says that there are no positive integers a, b, and c such that

$a^n + b^n = c^n$

for any values of n greater than 2.

1. *Write a function named* `check_fermat` *that takes four parameters—* `a`, `b`, `c` *and* `n`*—and that checks to see if Fermat's theorem holds. If n is greater than 2 and it turns out to be true that*
   $a^n + b^n = c^n$
2. *the program should print, "Holy smokes, Fermat was wrong!" Otherwise the program should print, "No, that doesn't work."*
3. *Write a function that prompts the user to input values for* `a`, `b`, `c` *and* `n`*, converts them to integers, and uses* `check_fermat` *to check whether they violate Fermat's theorem.*

Question 3

If you are given three sticks, you may or may not be able to arrange them in a triangle. For example, if one of the sticks is 12 inches long and the other two are one inch long, it is clear that you will not be able to get the short sticks to meet in the middle. For any three lengths, there is a simple test to see if it is possible to form a triangle:

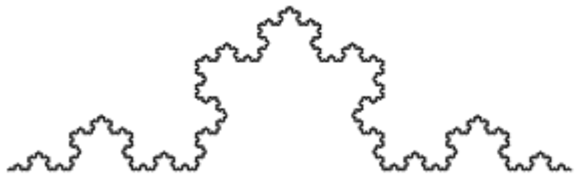
If any of the three lengths is greater than the sum of the other two, then you cannot form a triangle. Otherwise, you can. (If the sum of two lengths equals the third, they form what is called a "degenerate" triangle.)

1. *Write a function named* `is_triangle` *that takes three integers as arguments, and that prints either "Yes" or "No," depending on whether you can or cannot form a triangle from sticks with the given lengths.*
2. *Write a function that prompts the user to input three stick lengths, converts them to integers, and uses* `is_triangle` *to check whether sticks with the given lengths can form a triangle.*

Question 4

Read the following function and see if you can figure out what it does. Then run it

```
def draw(t, length, n):
    if n == 0:
        return
    angle = 50
    fd(t, length*n)
    lt(t, angle)
    draw(t, length, n-1)
    rt(t, 2*angle)
    draw(t, length, n-1)
    lt(t, angle)
    bk(t, length*n)
```



A Koch curve.

Question 5

The Koch curve is a fractal that looks something like the figure in Q9. To draw a Koch curve with length x, all you have to do is

1. *Draw a Koch curve with length x/3.*
2. *Turn left 60 degrees.*
3. *Draw a Koch curve with length x/3.*
4. *Turn right 120 degrees.*
5. *Draw a Koch curve with length x/3.*
6. *Turn left 60 degrees.*
7. *Draw a Koch curve with length x/3.*

The exception is if x is less than 3: in that case, you can just draw a straight line with length x.

1. *Write a function called `koch` that takes a turtle and a length as parameters, and that uses the turtle to draw a Koch curve with the given length.*
2. *Write a function called `snowflake` that draws three Koch curves to make the outline of a snowflake.*

Question 6

Write a function `is_between(x, y, z)` *that return* `True` *if* $x \le y \le z$ *or* `False` *otherwise.*

SECTION C

Reference: Think Python: How to Think Like a Computer Scientist by Allen B. Downey

temporary variable:
A variable used to store an intermediate value in a complex calculation.

dead code:
Part of a program that can never be executed, often because it appears after a `return` statement.

None:
A special value returned by functions that have no return statement or a return statement without an argument.

incremental development:
A program development plan intended to avoid debugging by adding and testing only a small amount of code at a time.

Question 1

The Ackermann function, A(m, n), is defined:

$$A(m, n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1, 1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m, n-1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

Write a function named `ack` that evaluates Ackermann's function. Use your function to evaluate `ack(3, 4)`, which should be 125. What happens for larger values of `m` and `n`?

Question 2

A palindrome is a word that is spelled the same backward and forward, like "noon" and "redivider". Recursively, a word is a palindrome if the first and last letters are the same and the middle is a palindrome.

The following are functions that take a string argument and return the first, last, and middle letters:

```
def first(word):
    return word[0]

def last(word):
    return word[-1]

def middle(word):
    return word[1:-1]
```

1. *Type these functions into a file named `palindrome.py` and test them out. What happens if you call `middle` with a string with two letters? One letter? What about the empty string, which is written `''` and contains no letters?*
2. *Write a function called `is_palindrome` that takes a string argument and returns `True` if it is a palindrome and `False` otherwise. Remember that you can use the built-in function `len` to check the length of a string.*

Question 3

A number, a, is a power of b if it is divisible by b and a/b is a power of b. Write a function called `is_power` that takes parameters `a` and `b` and returns `True` if `a` is a power of `b`. Note: you will have to think about the base case.

Reference: Think Python: How to Think Like a Computer Scientist by Allen B. Downey