1 Fruitful functions

A **fruitful function** is a function that returns a value when it is called. Most of the builtin functions that we have used are fruitful. For example, the function `abs` *returns* a new number – namely, the absolute value of its argument:

```
>>> abs(-42)
42
```

Some functions are *not* fruitful. For example, suppose `franklin` refers to a Turtle object, the function call `franklin.forward(100)` does *not* return anything. Instead it causes franklin to move forward. In addition, *all* of the functions defined in the previous handout are *not* fruitful.

2 Defining a fruitful function

If we want a function to return a result to the caller of the function, we use the `return` statement. For example, here we define two fruitful functions. The second one, `circle_area`, calls the first one, `square`, to square the radius.

```
import math

def square(x):
    return x * x

def circle_area(diameter):
    radius = diameter / 2.0
    return math.pi * square(radius)
```

In general, a return statement consists of the `return` keyword followed by an expression, which is evaluated and returned to the function caller.

How python evaluates a return statement Python evaluates fruitful functions pretty much the same way as non-fruitful ones (see last handout). The only thing new is how it executes a `return` statement. Here's roughly how it works:

1. Evaluate the expression to produce a data value.
2. Pass that data value back to the caller.
3. Leave the function *immediately* and return to the location where the function was called.

Python returns immediately when it reaches a `return` statement. The `print` statement in the function body below will *never* be executed:

```
def square(x):
    return x * x                  # python leaves function here...
    print "I am NEVER printed!"   # ... and never gets to here.
```

3 Return vs. print, which one to use?

Many beginning programmers get confused about the difference between **return** and **print**. The **return** statement sends a data value back to the caller of the function; the **print** statement displays a data value on the screen. You will need to use *both*. But when to use which? Here's a general rule of thumb: *Use return to share the result with another part of your program; use print to share the result with the user.*

The square function *must* use a **return** statement because it is called inside the circle_area function. In other words, the point of this function is to share the result (i.e., the squared number) with another part of our program (i.e., the circle_area function).

4 NoneType error

In python, *every* function is fruitful even if the body does not contain a **return** statement. In this case, the function will return a special value called None, which has type NoneType.

Example: suppose we change the **return** in square to **print**. Here's what happens when we call circle_area passing in 4 as its argument.

```
>>> circle_area(8)
16.0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "fruitful_functions.py", line 8, in circle_area
    return math.pi * square(radius)
TypeError: unsupported operand type(s) for *: 'float' and 'NoneType'
```

We get an error about a NoneType because circle_area is trying to multiply math.pi with the return value of square, which is None, and so it crashes. Also notice that just before the error, the program *prints* the number 16.0 (which is $(8/2)^2$).

5 Exercises

Solutions are presented in class and also included in the moodle version of this handout.

1. Write a function that computes the area of a ring with a given diameter and thickness.

Solution:

```
import math

def circle_area(diameter):
    radius = diameter / 2.0
    return math.pi * radius ** 2

# first of two solutions: this one is better
def area_ring(diameter, thickness):
    # area of a ring is the area of the outer
    # circle minus the area of the inner circle
    area_outer = circle_area(diameter)
    area_inner = circle_area(diameter - 2*thickness)
    return area_outer - area_inner

# second solution: this one is correct but bad because
# it calculates the area of the circles rather than
# call the circle_area function.  Don't repeat yourself!
def area_ring(diameter, thickness):
    area_outer = math.pi * (diameter/2) ** 2
    area_inner = math.pi * (diameter/2 - thickness) ** 2
    return area_outer - area_inner
```

2. Write a function, `max_of_two`, that takes two numbers and returns the larger number.

Solution: See next solution.

3. Write a function, `max_of_three`, that takes three numbers and returns the largest number.

Solution:

```
# MAX OF TWO
# version 1: use if to figure out larger
def max_of_two(a, b):
    if a > b:
        return a
    else:
        return b

# version 2: just use builtin max!
def max_of_two(a, b):
    return max(a, b)

# MAX OF THREE: three versions of max_of_three
# First version is correct but bad design.
# Second two versions are better because they
```

```python
# reuse existing functions. Obviously, in this
# case, python already has a builtin max function
# which can take three arguments, so using it is
# the best option!

# three versions: version 1 (bad)
def max_of_three(a, b, c):
    if a > b and a > c:
        return a
    elif b > a and b > c:
        return b
    else:
        return c

# three versions: version 2 (better)
def max_of_three(a, b, c):
    return max_of_two(max_of_two(a, b), c)

# three versions: version 3 (best!)
def max_of_three(a, b, c):
    return max(a, b, c)
```