CSCA08 Recipe for Designing Functions

This is the recipe that we will follow regularly in CSCA08 to design functions. At first it may seem like more work to write your functions following this recipe, but in the end it will save you time.

1. **Examples** Pick a name for the function (often a verb or verb phrase). Sometimes a good name is a short answer to the question "What does your function do?"

   Write one or two examples of calls to your function[1] and the expected returned values. Include an example of a *standard* case (as opposed to a tricky or corner case). Put the examples inside a triple-quoted string that you've indented since it will be the beginning of the docstring.

   ```
   """
   >>> is_even(2)
   True
   >>> is_even(17)
   False
   """
   ```

2. **Header** Write the function header above the docstring and outdent it. Choose a meaningful name for each parameter (often nouns). Include the *type contract* (the types of the parameters and return value).

   ```
   def is_even(value: int) -> bool:
       """
       >>> is_even(2)
       True
       >>> is_even(17)
       False
       """
   ```

3. **Description** Before the examples, add a description of what the function does and mention each parameter by name. Describe the return value.

   ```
   def is_even(value: int) -> bool:
       """Return True if and only if value is divisible by 2.

       >>> is_even(2)
       True
       >>> is_even(17)
       False
       """
   ```

4. **Body** Write the body of the function and indent it to match the docstring. To help yourself write the body, review your example cases from step 1 and consider how you determined the return values. You may find it helpful to write a few more example calls.

   ```
   def is_even(value: int) -> bool:
       """Return True if and only if value is divisible by 2.

       >>> is_even(2)
       True
       >>> is_even(17)
       False
       """

       return value % 2 == 0
   ```

5. **Test Your Function** Test your function on all your example cases including any additional cases you created in step 4. Additionally, try it on extra *tricky* or *corner* cases.

[1] Do not include examples for functions that involve randomness or input or output.

Another Example *Write a function that accepts the number of pizzas that you are ordering and the number of slices per pizza, and returns the total number of slices in the order.*

1. **Examples**

```
    """
    >>> total_slices(1, 8)
    8
    >>> total_slices(3, 12)
    36
    """
```

2. **Header**

```
def total_slices(num_pizzas: int, slices_per_pizza: int) -> int:
    """
    >>> total_slices(1, 8)
    8
    >>> total_slices(3, 12)
    36
    """
```

3. **Description**

```
def total_slices(num_pizzas: int, slices_per_pizza: int) -> int:
    """Return the total number of slices in num_pizzas pizzas that each have
    slices_per_pizza slices.

    >>> total_slices(1, 8)
    8
    >>> total_slices(3, 12)
    36
    """
```

4. **Body**

```
def total_slices(num_pizzas: int, slices_per_pizza: int) -> int:
    """Return the total number of slices in num_pizzas pizzas that each have
    slices_per_pizza slices.

    >>> total_slices(1, 8)
    8
    >>> total_slices(3, 12)
    36
    """

    return num_pizzas * slices_per_pizza
```

5. **Test** Call your function and compare the return values to what you are expecting.