CSC108 Recipe for Designing Functions (adapted for CSC148)

1. **Example** Write some 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 an indented triple-quoted string.

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

2. **Type Contract** Write a type contract that identifies name and type of each parameter. (See below for the notation to use for other types.) Choose a meaningful name for each parameter. Also identify the return type of the function. Put the type contract above the examples.

```python
@type num: int
@rtype: bool
>>> is_even(2)
True
>>> is_even(17)
False
```

3. **Header** Write the function header above the docstring and outdent it.

```python
def is_even(num):
    @type num: int
    @rtype: bool
    >>> is_even(2)
    True
    >>> is_even(17)
    False
```

4. **Description** In the same line as the opening triple-quote mark, put a one-line summary of what the function does. If necessary, you can put an optional, longer description above the type contract. Mention each parameter by name. If arguments to the function must satisfy any requirements beyond the type contract, specify them and label them as “Preconditions”.

```python
def is_even(num):
    """Return whether <num> is evenly divisible by 2.

    Preconditions: This function has none, but this is where they would go."""
```
5. **Body** Write the body of the function by remembering to indent it to match the docstring. To help yourself write the body, review your example cases from step 1 and how you determined the return values. You may find it helpful to write a few more example calls in the docstring.

```python
def is_even(num):
    """Return whether <num> is evenly divisible by 2.
    @type num: int
    @rtype: bool
    >>> is_even(2)
    True
    >>> is_even(17)
    False
    """
    return num % 2 == 0
```

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

---

**Notation for types**

Pycharm uses the following notation for the types in a type contract:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foo</td>
<td>Class Foo</td>
</tr>
<tr>
<td>Foo</td>
<td>Bar</td>
</tr>
<tr>
<td>(Foo, Bar)</td>
<td>Tuple of Foo and Bar</td>
</tr>
<tr>
<td>list[Foo]</td>
<td>List of Foo elements</td>
</tr>
<tr>
<td>dict[Foo, Bar]</td>
<td>Dict from Foo to Bar</td>
</tr>
<tr>
<td>(Foo, Bar) -&gt; Baz</td>
<td>Function of Foo and Bar that returns Baz</td>
</tr>
<tr>
<td>list[dict[str, datetime]]</td>
<td>List of dicts from str to datetime</td>
</tr>
</tbody>
</table>

This is a slightly simplified version of the table published at: