Section B2 Refactoring & Test Driven Development (TDD)

Refactoring

Use boolean values directly

Making comparisons like 'if a == True' can be replaced by directly using the boolean value.

```
# Not this:
if a == True:
  b = False
else:
  b = True
print (a, b)

# Instead refactor:
b = not a
print (a, b)
```

Use list comprehensions

A more concise, readable and pythonic approach is to use list comprehensions. Quite often this is an easy refactoring approach to apply.

```
# Rather than this loop:
h_letters = []

for letter in 'human':
    h_letters.append(letter)

print(h_letters)

# Refactor like:
h_letters = [letter for letter in 'human']
print( h_letters)

# it's also possible to apply a conditional statement when using list comprehensions
vowels = ['a', 'e', 'i', 'o', 'u']
h_letters = [letter for letter in 'human' if letter in vowels]
print( h_letters)
```

Use any() instead of a loop

Rather than iterate through a list, we can use the any() function and a list comprehension to quickly and pythonically determine if any item in a list conforms to a boolean condition.

```
numbers = [-1, -2, -4, 0, 3, -7]
has_positives = False
for n in numbers:
    if n > 0:
        has_positives = True
        break

# Instead refactor
has_positives = any(n > 0 for n in numbers)
```

Replace a manual loop counter with enumerate()
Within a loop, it's common to want to know which is the current iteration - this can be done with a list counter. A better alternative is to let Python keep track of the iteration... it has to anyway! ;-)

```
# Manually counting...
i = 0
for player in players:
    print(i, player)
    i += 1

# Instead refactor:
for i, player in enumerate(players):
    print(i, player)
```

Use default values on functions where appropriate

```
# Not this:
def maybe_repeat (thing, number_of_times):
  # do some stuff in a loop
  pass

maybe_repeat (my_thing, 1)

# Instead refactor:
def maybe_repeat (thing, number_of_times = 1):
  # do some stuff in a loop just once - or more if required
  pass

maybe_repeat (my_thing)
```

Test Driven Development (TDD)

Red-Green-Refactor and Test Harnesses
Test Driven Development is an iterative process that is often automated with a supporting test harness or test framework. Building a suite of tests with known inputs and expected outputs is fundamental to TDD and the heart of the the

"*Red-Green-Refactor*" methodology so called for its three phases:

1. write a *failing test* for a requirement,
2. implement just enough to pass the test for that requirement, and
3. refactor code if required to improve the quality of the implementation (and retest!).