COMP1730/COMP6730
Programming for Scientists

Examples on Code Quality & Debugging

Code quality example

Lab 4 has a debugging question about a function that returns the position of an element in a sequence. Here is an implementation:

```
def find_element(sq, y):
    x = 0 # index
    # this handles the case when y is not in the sequence
    if len(sq) == 0:
        return 0
    while sq[x] != y:
        x = x + 1
        # np.max returns the maximum number in an array
        if x < len(sq):
            x = x + 1 – 1 # don't want to change i again
        else:
            return x
    # this is the end of the loop
    return x
```

Can the quality of this code be improved?

Aspects of code quality: reminder

1. Docstrings: Descriptions of the purpose, inputs, outputs, assumptions?
2. Variable and function naming: good and descriptive?
3. Comments: appropriate/relevant?
4. Code organisation: non-redundant? easy to understand? functional decomposition?
5. Code efficiency when necessary.

1. Docstrings

```
def find_element(sq, y):
    """Return the position of element y in the sequence sq.
    If y is not present, return the sequence length.
    """
    x = 0 # index
    # this handles the case when y is not in the sequence
    if len(sq) == 0:
        return 0
    while sq[x] != y:
        x = x + 1
        # np.max returns the maximum number in an array
        if x < len(sq):
            x = x + 1 - 1 # don't want to change i again
        else:
            return x
    # this is the end of the loop
    return x
```

2. Naming

* Rename `sq` to `seq` or `sequence`.
* Rename `y` to `target` or `target_element` (`x` and `y` could be used for coordinates).
* Rename `x` to `i` or `index`.

Code after name refactoring

```
def find_element(seq, target):
    """Return the position of element target in the sequence seq.
    If target is not present, return the sequence length.
    """
    i = 0 # index
    # this handles the case when target is not in the sequence
    if len(seq) == 0:
        return 0
    while seq[i] != target:
        i = i + 1
        # np.max returns the maximum number in an array
        if i < len(seq):
            i = i + 1 - 1 # don't want to change i again
        else:
            return i
    # this is the end of the loop
    return i
```

3. Comments

* Wrong comment:

```
    # this handles the case when target is not in the sequence
    if len(seq) == 0:
        return 0
```

* Better change to:

```
    # handles special case of empty sequence
    if len(seq) == 0:
        return 0
```

Code after adjusting comments

```
def find_element(seq, target):
    """Return the position of element target in the sequence seq.
    If target is not present, return the sequence length.
    """
    i = 0
    # handles special case of empty sequence
    if len(seq) == 0:
        return 0
    # going through positions of input sequence to find the element
    while seq[i] != target:
        i = i + 1
        if i < len(seq):
            i = i + 1 - 1 # don't want to change i again
        else:
            return i
    # target is found at position i
    return i
```

4. Code organisation

* Unnecessary code:

```
if i < len(seq):
    i = i + 1 – 1 # don't want to change i again
else:
    return i
```

* Better change to:

```
if i >= len(seq):
    return i # reaching the end, target is not found
```

Together after improvements

```
def find_element(seq, target):
    """Return the position of element target in the sequence seq.
    If target is not present, return the sequence length.
    """
    i = 0
    # handles special case of empty sequence
    if len(seq) == 0:
        return 0
    # going through positions of input sequence to find the element
    while seq[i] != target:
        i = i + 1
        if i >= len(seq):
            return i # reaching the end, target is not found
    # target is found at position i
    return i
```

However, it takes time to understand how the while loop works.

So better:

```
def find_element(seq, target):
    """Return the position of element target in the sequence seq.
    If target is not present, return the sequence length.
    """
    # going through positions of input sequence to find the element
    for index in range(len(seq)):
        if seq[index] == target:
            return index  # target is found
    # target is not found
    return len(seq)
```

Debugging example

Lab 4 has an exercise to debug an incorrect code (assuming number is non-negative):

```
def sum_even_digits(number):
    m = 1 # the position of the next digit
    dsum = 0 # the sum
    while number % (10 ** m) != 0:
        # get the m:th digit
        digit = (number % (10 ** m)) // (10 ** (m - 1))
        # only add it if even:
        if digit % 2 == 0:
            dsum = dsum + digit
        m = m + 1
    return dsum
```

How to debug?

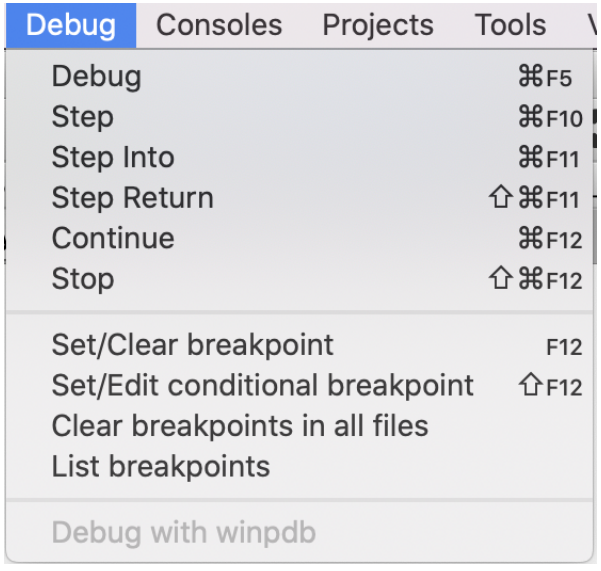
1. Test: Design some test cases and run through test(s) where the code fails.
2. Locate the line(s) that caused the bug:
   - Use `print` statement, esp. inside a loop.
   - Use debugging facility of the IDE.
3. Correct the code without introducing new bug.

1. Testing

```
# edge cases
assert sum_even_digits(0) == 0
assert sum_even_digits(1) == 0

# Only odd digits
assert sum_even_digits(1537) == 0

# Only even digits
assert sum_even_digits(2604) == 2+6+4

# Mixed odd and even digits:
assert sum_even_digits(25048) == 14 # first and last digits are even
assert sum_even_digits(32059) == 2 # first and last digits are odd
assert sum_even_digits(5470) == 4 # first odd digit, last even digit
assert sum_even_digits(61123) == 8 # first even digit, last odd digit
```

Live demo now

Debugging in Spyder



* **Debug**: start a debug session
* **Step**: Run current line
* **Step Into**: Go into a function at current line
* **Step Return**: Run until current function returns
* **Continue**: Run until the next breakpoint
* **Stop**: Stop debug session

Take home messages

* Remember important aspects of coding quality: docstrings, naming, comments, and code organisation.
* Try to utilise the debugging facility of the IDE (e.g. Spyder).