LEIDEN UNIVERSITY MEDICAL CENTER

Good Practices for Programming

Jeroen F. J. Laros
Leiden Genome Technology Center
Department of Human Genetics
Center for Human and Clinical Genetics

Introduction

A very short and incomplete introduction to good practices for programming.

We are trying to avoid:

- Throw away code.
- Reinventing the wheel.
- Wasting time reading your own code.
- Wasting time reading *someone else's* code.
- ...

We will try to show the use and advantages of some standard practices by giving examples.

Introduction

We shall *not* cover software development as a whole:

- Requirements engineering.
- Design.
- Planning.
- Releasing.
- ...

We focus on basic programming.

Introduction

We *will* focus on:

- Indentation.
- Documentation.
  - Inline comment.
  - Choosing good names.
- Functions.
  - Parameters and return values.
  - Documentation conventions.
- Unit tests.

Indentation

Indentation is a way to reveal the control flow of a program.

```
1       for i in range(10) :
2           print i                     # Executed 10 times.
3           if i == 1 :
4               print "hey"             # Executed if i equals 1.
5       #for
```

Listing 1: Indented code

Line 2, 3 and 4 are in the *scope* of the first **for**-loop.

Line 4 is in the scope of the **if**-clause.

Indentation

A piece of C code without indentation.

```
1    for (i = 0; i < 10; i++)
2    for (j = 0; j < 20; j++) {
3    if (i == j)
4    printf("Equal\n");
5    if (i > j)
6    printf("Larger\n");
7    if (i < j)
8    printf("Smaller\n");
9    }
```

Listing 2: Unindented code

C (unlike Python) does not *require* indentation.

Indentation

A piece of C code without indentation.

```
1   for (i = 0; i < 10; i++) {
2   for (j = 0; j < 20; j++)
3   if (i == j)
4   printf("Equal\n");
5   if (i > j)
6   printf("Larger\n");
7   if (i < j)
8   printf("Smaller\n");
9   }
```

Listing 3: Unindented code

Notice the small change: the { from line 2 is now on line 1.

Indentation

Use indentation to reveal the control flow of the code.

```
1   for (i = 0; i < 10; i++)
2     for (j = 0; j < 20; j++) {
3       if (i == j)
4         printf("Equal\n");
5       if (i > j)
6         printf("Larger\n");
7       if (i < j)
8         printf("Smaller\n");
9     }
```

Listing 4: Same code as in Listing 2

All **if** statements are in the scope of the inner **for**-loop.

Indentation

Notice the same small change, it is easier to spot.

```
1   for (i = 0; i < 10; i++) {
2     for (j = 0; j < 20; j++)
3       if (i == j)
4         printf("Equal\n");
5     if (i > j)
6       printf("Larger\n");
7     if (i < j)
8       printf("Smaller\n");
9   }
```

Listing 5: Same code as in Listing 3

Only the first **if** statement is in the scope of the inner **for**-loop.

Functions

Functions (procedures, methods, subroutines) are used for:

- Code reuse.
- Structuring code.
- Abstraction.
- ...

In general, try to use functions as much as possible

Functions

A piece of sample code (with an intentional flaw).

```
1    c1 = a1 * a1
2    c2 = b1 * b1
3    d1 = c1 + c2 / 2
4
5    c1 = a2 * a2
6    c2 = b2 * b2
7    d2 = c1 + c2 / 2
8
9    c1 = a3 * a3
10   c2 = b3 * b3
11   d3 = c1 + c2 / 2
12
13   c1 = a4 * a4
14   c2 = b4 * b4
15   d4 = c1 + c2 / 2
```

Listing 6: Flat, repetitive code

Functions

Writing comment often helps to identify code duplication.

```
1   c1 = a1 * a1      # c1 is a1^2.
2   c2 = b1 * b1      # c2 is b1^2.
3   d1 = c1 + c2 / 2 # d1 is the average of c1 and c2.
4
5   c1 = a2 * a2      # Same as previous block,
6   c2 = b2 * b2      # but now fow a2 and b2.
7   d2 = c1 + c2 / 2
8
9   c1 = a3 * a3      # Same as previous block,
10  c2 = b3 * b3      # but now for a3 and b3.
11  d3 = c1 + c2 / 2
12
13  c1 = a4 * a4      # Same as previous block,
14  c2 = b4 * b4      # but now for a4 and b4.
15  d3 = c1 + c2 / 2
```

Listing 7: Inline documentation

Functions

Use functions to avoid code duplication.

```
1   def f(a, b) :
2       c1 = a * a          # c1 is a1^2
3       c2 = b * b          # c2 is b1^2
4
5       return c1 + c2 / 2 # The average of c1 and c2.
6   #f
7
8   d1 = f(a1, b1)
9   d2 = f(a2, b2)
10  d3 = f(a3, b3)
11  d4 = f(a4, b4)
```

Listing 8: Same functionality as in Listing 7

It is now trivial to add more calls to this function.

Functions

Functions also makes altering code easier.

```
1   def f(a, b) :
2       c1 = a * a * a      # c1 is a1^3
3       c2 = b * b * b      # c2 is b1^3
4
5       return c1 + c2 / 2 # The average of c1 and c2.
6   #f
7
8   d1 = f(a1, b1)
9   d2 = f(a2, b2)
10  d3 = f(a3, b3)
11  d4 = f(a4, b4)
```

Listing 9: Cubes instead of squares

Altering the same behaviour in the flat code would be time consuming and prone to errors.

Functions

Use meaningful names for functions and variables.

```
beer = 2 * yoghurt + bicycle
```

Listing 10: Funny, but uninformative

```
insertSize = 2 * readLength + internalSize
```

Listing 11: Same functionality as Listing 10

Names should reflect the function.

Functions

Use meaningful names for functions and variables.

```
1   def meanOfSquares(a, b) :
2       aSquare = a * a
3       bSquare = b * b
4
5       return aSquare + bSquare / 2
6   #meanOfSquares
7
8   d1 = meanOfSquares(a1, b1)
9   d2 = meanOfSquares(a2, b2)
10  d3 = meanOfSquares(a3, b3)
11  d4 = meanOfSquares(a4, b4)
```

Listing 12: Same functionality as Listing 8

Notice that inline comment is no longer needed (in this case).

Functions

Document each function.

```
1   def meanOfSquares(a, b) :
2       """
3       Calculate the mean of the squares of two numbers.
4
5       Arguments:
6           int a ; The first number.
7           int b ; The second number.
8
9       Returns:
10          int ; Mean of the squares.
11      """
12
13      aSquare = a * a
14      bSquare = b * b
15
16      return aSquare + bSquare / 2
17  #meanOfSquares
```

Listing 13: Documented function

Functions

Alternative documentation.

```
1    def meanOfSquares(a, b) :
2        """
3        Calculate the mean of the squares of two numbers.
4
5        @arg a: The first number.
6        @type a: integer
7        @arg b: The second number.
8        @type b: integer
9
10       @returns: Mean of the squares.
11       @rtype: integer
12       """
13
14       aSquare = a * a
15       bSquare = b * b
16
17       return aSquare + bSquare / 2
18   #meanOfSquares
```

Listing 14: Documentation with a markup language

Unit tests

Test the implementation of a function.

```
1    def testMeanOfSquares() :
2        """
3        Test the meanOfSquares() function.
4        """
5
6        assert meanOfSquares(1, 1) == 1
7        assert meanOfSquares(2, 4) == 10
8    #testMeanOfSquares
```

Listing 15: Unit test of Listing 14

This function will report an error because of a flaw in the function.

Functions

Because of the unit test, we identified the flaw.

```
1   def meanOfSquares(a, b) :
2       """
3       Calculate the mean of the squares of two numbers.
4
5       @arg a: The first number.
6       @type a: integer
7       @arg b: The second number.
8       @type b: integer
9
10      @returns: Mean of the squares.
11      @rtype: integer
12      """
13
14      aSquare = a * a
15      bSquare = b * b
16
17      return (aSquare + bSquare) / 2
18  #meanOfSquares
```

Listing 16: Final function

Questions?