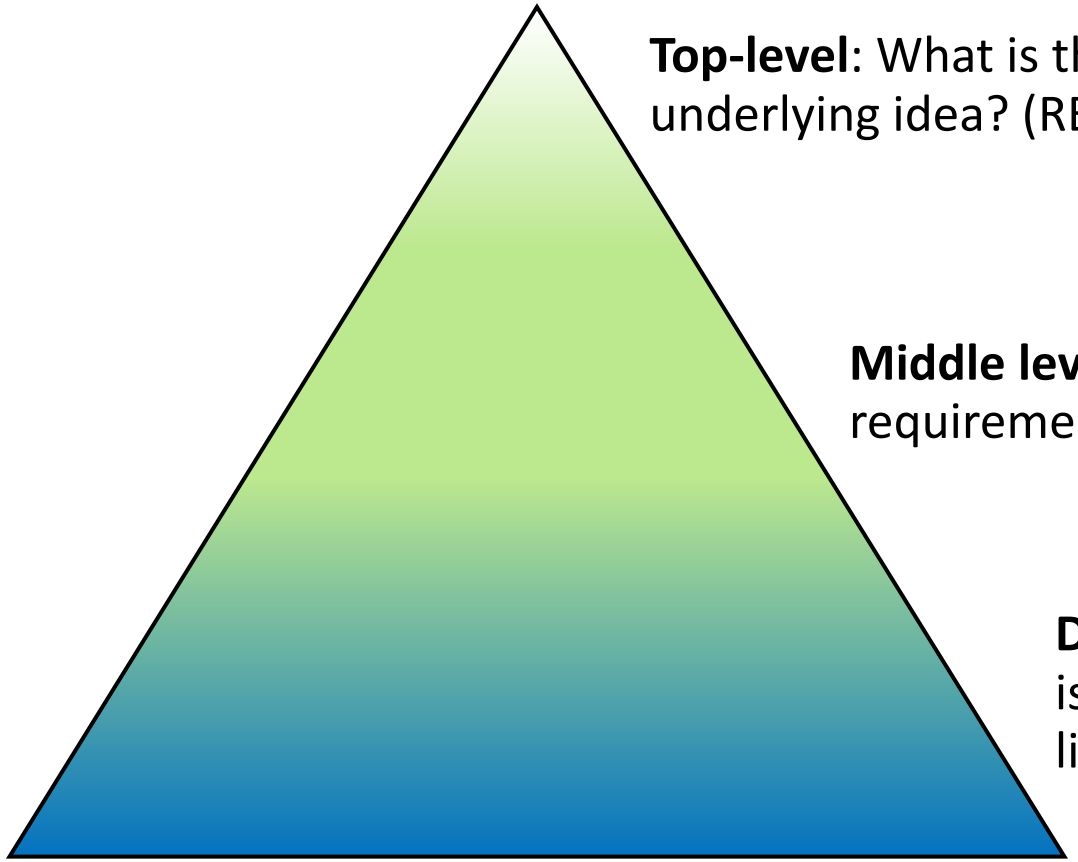
Bridging the gaps in bioinformatics/Getting started in bioinformatics

Why and how to document your code

Documentation can be done at multiple levels



Top-level: What is this code for? What is the underlying idea? (README/Wiki)

Middle level: How can I use it? (Usage info, installation requirements, scope, options)

Deep level: How does the code work? What is the purpose of each chunk? (Comment lines and structure)

Why should you care about documentation?

Transparency

- Colleagues using the code will be able to point out issues, if they understand what the code is doing
- Colleagues with other expertise (wetlab, epidemiology etc) may also be able to point out issues you are not aware of

Why should you care about documentation?

Transparency

- Colleagues using the code will be able to point out issues, if they understand what the code is doing
- Colleagues with other expertise (wetlab, epidemiology etc) may also be able to point out issues you are not aware of

Collaboration

- Give your colleagues a chance to contribute and improve on the code
- Give your colleagues a chance to take over from you

Why should you care about documentation?

Transparency

- Colleagues using the code will be able to point out issues, if they understand what the code is doing
- Colleagues with other expertise (wetlab, epidemiology etc) may also be able to point out issues you are not aware of

Collaboration

- Give your colleagues a chance to contribute and improve on the code
- Give your colleagues a chance to take over from you

Reproducility

- This is how we do science!

Why should you care about documentation?

Transparency

- Colleagues using the code will be able to point out issues, if they understand what the code is doing
- Colleagues with other expertise (wetlab, epidemiology etc) may also be able to point out issues you are not aware of

Collaboration

- Give your colleagues a chance to contribute and improve on the code
- Give your colleagues a chance to take over from you

Reproducility

- This is how we do science!

Helping yourself out

- It is surprisingly easy to forget the details of code you wrote a while ago..

README!

- Format of README file is not fixed. Content depends on your project
- Start with overview, then go more detailed further down.
- The worst README is the one that doesn’t exist!

Update your README file continously while you develop your code!

Usage information

Well-written code has integrated usage information.

- Something useful should happen if you execute the software without arguments (or with the -h or --help flag).

```
[redacted]$ bcftools

Program: bcftools (Tools for variant calling and manipulating VCFs and BCFs)
License: GNU GPLv3+, due to use of the GNU Scientific Library
Version: 1.14 (using htslib 1.14)

Usage:   bcftools [--version|--version-only] [--help] <command> <argument>

Commands:

 -- Indexing
    index        index VCF/BCF files

 -- VCF/BCF manipulation
    annotate     annotate and edit VCF/BCF files
    concat       concatenate VCF/BCF files from the same set of samples
    convert      convert VCF/BCF files to different formats and back
    isec         intersections of VCF/BCF files
```

Script level documentation

- Keep a clear and logical structure.
- Name your variables properly
- Use comments (#)

get_ambiguous_positions.py

```
#!/usr/bin/env python3
import sys
import os
from Bio import SeqIO
from Bio.Seq import Seq


#Usage: python length_fasta.py sequences.fasta

#List of characters, which will not be counted as ambiguous
non_amb = ["A","C","T","G","a","c","t","g","N","n","-"]

#Print a header of output
print("Seq_id\tPosition\tAmb")

#Read in fasta-file, and loop over sequences (first "for loop")
fasta_file = sys.argv[1]
for seq_record in SeqIO.parse(fasta_file, "fasta"):
    #Get sequence id, sequence and sequence length
    seq_id = seq_record.id
    seq_str = str(seq_record.seq)
    seq_length = len(seq_str)
    #Loop over each character in the sequence, and check if it exists in the "non_amb" list. If not, print sequence-id, position, and character
    position=1
    for character in seq_str:
        if character not in non_amb:
            print(seq_id, position, character, sep="\t")
        position = position + 1 #remember to add 1 to the position variable, as you move through the sequence string. Also when the character is not ambiguous!
```

Code evolves

You write a script, and all is well. But then:

- There is a bug
- You need another usage option
- Or an additonal script
- There is a dependency that got updated
- Change in wetlab

You write a script, and all is well. But then:

- Turns out that last add-on was a really bad idea. Must be changed. Meanwhile, users should go back to the original version. Did you save it somewhere?

=> **You need version control software**

Acknowledgements

The creation of this training material was commissioned by ECDC to Statens Serum Institut (SSI) with the direct involvement of Kirsten Ellegaard