How to read code

Terence Parr
MSDS program
University of San Francisco

Why must we be able to read code?

- Fundamentally, programmers communicate with code
- It's how we express our thoughts to the computer and also to other developers
- It's how we learn from others and improve our skills; example:

Student was doing

```
df['A'] = False
df.loc[(df['type']=='foo')&(df['alt_aa']==df['ref_aa']),'A']=True
```

Later, they saw this

```
df['A'] = (df['type']=='foo')&(df['ref_aa']==df['alt_aa'])
```

Reading code is also part of your process

- We can often find hints or solutions to a coding problem through code snippets found via Google search or at StackOverlow
- The code is the documentation
  - The complete behavior of a function isn't always clear from just the name or parameter list
  - Looking at the source code is the best way to understand what it does
- All code has bugs, particularly code we just wrote that has not been tested exhaustively
  - As part of the coding process, we are constantly bouncing around, reading our existing code base to make sure everything fits together

How to read code

- Our first clue comes from the fact that we are not computers, hence, we should not read code like a computer, examining one symbol after the other
- Instead, we look for key elements and code patterns
- We reverse the process followed by the code author
  - Code author thought "*convert prices into a new list by dividing by 2*," which they converted to "map" pattern and then to a Python loop
  - We must reverse this and imagine the original goal of author
  - Don't try to determine functionality by simulating the loops statements literally in your head or on paper
  - Rather, look for patterns that tell us what high-level operations are being performed

Good incentive to write clear code

- You should emphasize clarity when writing code, so that reading the code quickly leads the reader to your intentions
- The reader could be you or…
- There is an excellent quote (by John F. Woods I think) that summarizes things well:

> "*Always code as if the person who ends up maintaining your code will be a violent psychopath who knows where you live.*"

Getting the gist of a program

- When looking at a textbook for the first time, it makes sense to scan through the table of contents to get an overall view of the content
- Same for code: Look through all of the files and the names of the functions contained in those files
- Figure out where the main program is
- Depending on your purpose in reading the program, you might start stepping through the main program or immediately jump to a function of interest.
- Look at the input-output pairs of the program from sample runs or tests because it helps you understand the program's functionality

Getting the gist of a function

Look at the function name

- Once we identify a main program or function to examine, it's time to reverse-engineer the function work plan / algorithm
- The function's name is perhaps the biggest clue as to what the function does, assuming the code author was a decent programmer; e.g., there is no doubt what the following function's goal is:

```
def average(...):
    ...
```

- (Using a generic function name like **f** is how faculty write code-reading questions without giving away the answer.)

Function comments

- Sometimes programmers will provide comments about the usage of a function, but be careful!
- Programmers often change the code without changing the comments and so the comments will be misleading

Function parameters and return value

- The next step is to identify the parameters and return value(s)
- The names of the parameters often tell us a lot
- Unfortunately, Python usually does not have explicit types (they aren't checked by Python anyway) so we have to figure that out ourselves
- Knowing the types of values and variables is critical to understanding a function; without knowing the types, you cannot know what the function does

```
def average(data):
    ...
    return sum/n
```

Don't fall for this trick in interviews. W/o knowing the type, we don't know what the operators do

Reading the function code itself

- Scan the statements of the function *looking for loops*; all of the action occurs in the loops
- An inexperienced programmer examines the statements of the function individually and literally, emulating a computer to figure out the emergent behavior
- An experienced programmer looks for **patterns** in the code that are implementations of high-level ops like map, search, filter, etc...
- Analogy: consider memorizing the state of a chessboard in the middle of play
  - A beginner has to memorize where all of the pieces are individually
  - A chessmaster recognizes that the board is, say, merely a variation on the Budapest Gambit

Identify the pattern, fill in the holes

- What pattern is this code following? I.e., what's it doing?

```
sum = 0.0
for x in data:
    sum = sum + x
```

- It's using the accumulator pattern, "sum up a bunch of stuff"
- The "holes" are: what we are accumulating (data), the operation is summation; we might also have to look for loop bounds

Code sample

- I have deliberately used crappy variable names so you have to focus on the functionality:

```
foo = []
for blah in blort:
    foo.append(bar * 2)
```

- That's a "map" operation that translates one list to another
- The clue is initialization of empty list and loop around something that adds to the list as a function of bar; it's bar → foo

Code sample

- What high-level math operation is this performing?

```
for i in range(n):
    for j in range(n):
        C[i][j] = A[i][j] + B[i][j]
```

- The key is to read “nested loop” as all combinations of n x n
- Then look at the operation, which is adding two elements
- Putting it together, add two matrices
- See nested loops? Think matrix or image operations

Code sample

- Quick! What does this do?

```
blort = -99999
for x in X:
    if x > blort:
        blort = x
print(blort)
```

- Finds max value in x
- Anytime you see an if statement inside of a loop, think **filter** or **search** or **accumulator** with condition.

Code sample

- Describe what value bar has after this code completes

```
foo = []
bar = []
for blah in blort:
    foo.append(blah * 2)
for zoo in foo:
    if zoo>10:
        bar.append(zoo)
```

- Nothing more than two patterns in a sequence
- **Map** blort into foo then **filter** values in foo to get zoo

Summary

- Code is how we communicate; code **is** the documentation
- Reading code is about identifying patterns, reverse engineering the intent of the original programmer
- Be kind to other developers and your future self by writing high-quality code; and include useful comments
- That includes choosing excellent variable and function names and writing code that clearly illustrates your intent