# Reading Code, Debug

## Problem Solving using Python - Week 6

# Week 6 - Learning Objectives

# Week 6 - Learning Objectives

## You will be able to

# Week 6 - Learning Objectives

## You will be able to

1. ... read code in a methodical and thoughtful manner.

# Week 6 - Learning Objectives

## You will be able to

1. ... read code in a methodical and thoughtful manner.
2. ... *debug* code well.

# Motivation - Reading Code and Debug?

# How will we Learn to Read Code and to Debug?

# How will we Learn to Read Code and to Debug?

## Étude

# How will we Learn to Read Code and to Debug?

## Étude

> "An instrumental musical composition, usually short, of considerable difficulty, and designed to provide practice material for perfecting a particular musical skill." - Wikipeida

# Reading Code

# Why do we Read Code?

# Why do we Read Code?

1. Debugging a Program

# Why do we Read Code?

1. Debugging a Program
2. Modifying a Program

# Why do we Read Code?

1. Debugging a Program
2. Modifying a Program
3. Using a Program (Code as Documentation)

# Why do we Read Code?

1. Debugging a Program
2. Modifying a Program
3. Using a Program (Code as Documentation)
4. Learning (e.g., Worked Example, Stack Overflow)

# Why do we Read Code?

1. **Debugging a Program** - Étude!
2. Modifying a Program
3. Using a Program (Code as Documentation)
4. Learning (e.g., Worked Example, Stack Overflow)

# Let's Extract the Structure of a Natural Language Text

# Let's Extract the Structure of a Natural Language Text

> "(1) People say that a dog ”knows” its name (2) because it comes when it is called, and (3) that it ”remembers” its master, (4) because it looks sad in his absence, but (5) wags its tail and barks when he returns." - Bertrand Russell, The Analysis of Mind, Lecture I.

# Let's Extract the Structure of a Natural Language Text

> "(1) People say that a dog ”knows” its name (2) because it comes when it is called, and (3) that it ”remembers” its master, (4) because it looks sad in his absence, but (5) wags its tail and barks when he returns." - Bertrand Russell, The Analysis of Mind, Lecture I.
> "(1) The fact is that, as a rule, a specific protein is produced by a cell in very small quantities, sometimes a mere one or two molecules per cell. (2) As a result, the production of proteins needed for particular research becomes an arduous and costly undertaking. (3) One has to process dozens of kilograms, nay tons, of biomass to obtain milligrams of protein. (4) Despite such meager quantities, it is still not possible to ensure the necessary purity of the protein. (5) Hence, the costs of many protein preparations are exorbitant and their purity is substandard." - Maxim D. Frank-Kamenetskii, Unraveling DNA, trans. Lev Liapin (New York: VCH Publishers, 1993), 61.

# Reading Code - What is it all about?

# Reading Code - What is it all about?

- Program's Code = A Solution to a Programming Problem

# Reading Code - What is it all about?

- Program's Code = A Solution to a Programming Problem
- The goal is to extract the design and its implementation details, in some sense, performing **reverse engineering** of the code

# Reading Code - What is it all about?

- Program's Code = A Solution to a Programming Problem
- The goal is to extract the design and its implementation details, in some sense, performing **reverse engineering** of the code

> "When you do this exercise, think of yourself as an anthropologist, trucking through a new land with just barely enough of the local language to get around and survive. Except, of course, that you will actually get out alive because the internet isn't a jungle." - Zed Shaw in "Learn Python the Hard Way".

# The Two Questions to Answer

# The Two Questions to Answer

1. **What** is the *problem* that the program solves?

# The Two Questions to Answer

1. **What** is the *problem* that the program solves?

2. **How** does the problem solve the problem? What is the *design* of the code?

# Reading Code Steps

**The process of reading code echoes**

**the "Programming Problem Solving Model"**

# Reading Code Steps

1. Apply the *Reinterpret the Problem* Phase
2. Split the Code into Sections with Goals
3. Identify the Meaning of Each Variable
4. Generate Test Cases
5. Walk Through each Section

# Reading Code Steps

1. ➡ Apply the *Reinterpret the Problem* Phase
2. Split the Code into Sections with Goals
3. Identify the Meaning of Each Variable
4. Generate Test Cases
5. Walk Through each Section

# Reading Code Steps

1. Apply the *Reinterpret the Problem* Phase
2. ➡ Split the Code into Sections with Goals
3. Identify the Meaning of Each Variable
4. Generate Test Cases by the Test Phase
5. Walk Through each Section

1. Identify sections in the code
   - Spaces, flow controls (loops, conditions), functions
2. Identify goals for each section
3. Use the Comments

# Reading Code Steps

1. Apply the *Reinterpret the Problem* Phase
2. Split the Code into Sections with Goals
3. ➡ Identify the Meaning of Each Variable
4. Generate Test Cases by the Test Phase
5. Walk Through each Section

1. Variable Names
2. Look at the **Usage** of Each Variable
    1. Where the variable is **used**?
    2. Is the variable **modified**? **Where** is the variable modified (also defined)?
    3. *How* the variable is **used**?

# Reading Code Steps

1. Apply the *Reinterpret the Problem* Phase
2. Split the Code into Sections with Goals
3. Identify the Meaning of Each Variable
4. ➡ Generate Test Cases
5. Walk Through each Section

# Reading Code Steps

1. Apply the *Reinterpret the Problem* Phase
2. Split the Code into Sections with Goals
3. Identify the Meaning of Each Variable
4. Generate Test Cases
5. ➡ Walk Through each Section

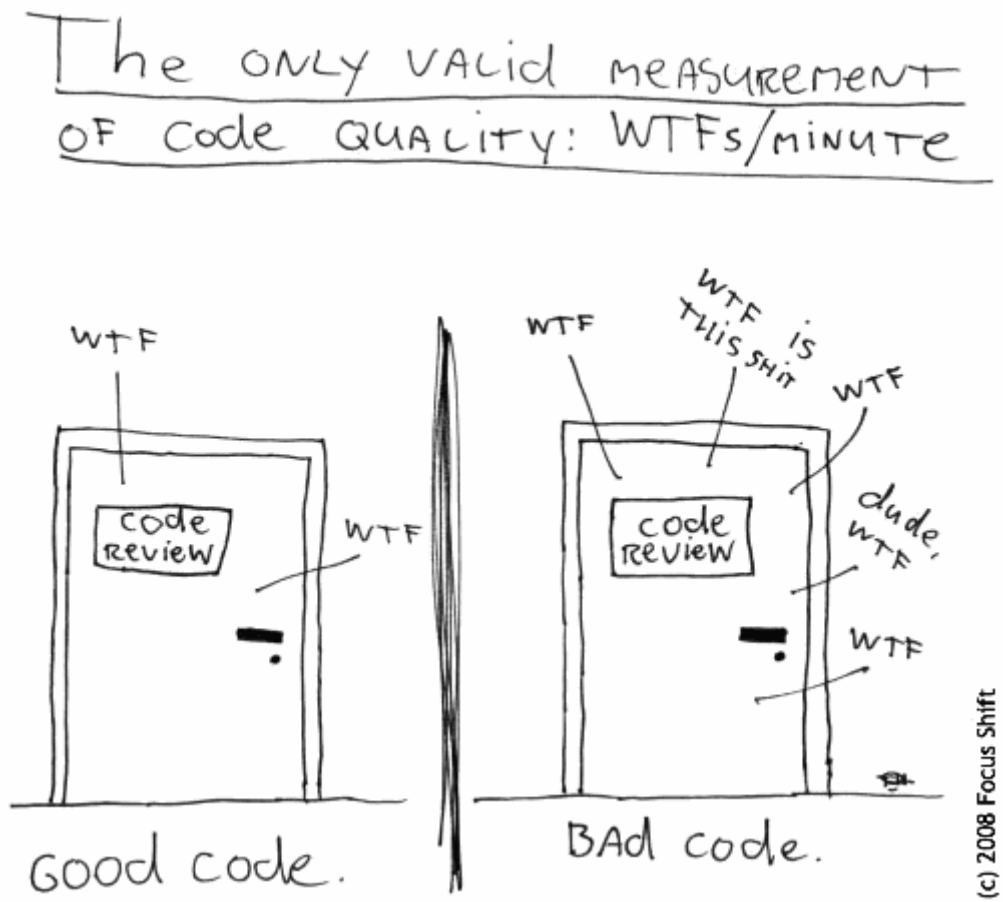
1. Follow Execution
2. Track Variables
    1. Think-aloud
    2. Written
3. Note to Indentation
4. Pay Attention in Conditions and Loops

# Reading Code Steps

1. Apply the *Reinterpret the Problem* Phase
2. Split the Code into Sections with Goals
3. Identify the Meaning of Each Variable
4. Generate Test Cases
5. Walk Through each Section

# Reading Code
# Wrap-up + Q&A

(this is not the end yet)


The ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE
WTF
code review
WTF
!
Good code.
WTF
WTF is this shit
WTF
code review
dude, WTF
WTF
!
BAd code.
(c) 2008 Focus Shift

# Debug Phase

# Programming Problem Solving Model

1. Reinterpret the Problem
2. Design a Solution
3. Code
4. Test
5. Debug
6. Evaluate & Reflect

# How to Avoid Debugging?

**"Everyone knows that debugging is twice as hard as writing a program in the first place.**

**So if you're as clever as you can be when you write it, how will you ever debug it?"**

**Brian Kernighan in "The Elements of Programming Style" (1974)**

# How to Avoid Debugging?

1. Write **tests** at the Problem phase (e.g. using `asserts`)
2. Use **incremental development** = Get something working and keep it working
    1. Start small
    2. Keep it working

**Problem**  **Solution**

`assert`

🎉 🎊

# How do you Debug?

# How Debugging should not Look Like?

# Why is Debugging Hard?

## Programmer

## Program

# Why is Debugging Hard?

**Programer** Mental Gap **Program**

# How to Debug? - The Fundamental Mindset

# How to Debug? - The Fundamental Mindset

## The Scientific Method

# How to Debug? - The Fundamental Mindset

## The Scientific Method

Think Like ...

**Empirical Scientist** | **Detective** | **Doctor**

# How to Debug? - The Fundamental Mindset

## The Scientific Method

Think Like ...

**Empirical Scientist** | **Detective** | **Doctor**

Debugging Systematically

# How to Debug? - The Fundamental Mindset

## The Scientific Method

Think Like ...

**Empirical Scientist** | **Detective** | **Doctor**

Debugging Systematically

Everyone is a suspect (Except Python)

# The Four Questions to Answer

# The Four Questions to Answer

1. **What** is the bug?
2. **Why** does the bug happen? What is the cause of the bug? What is the **root cause**?
3. **Where** in the code is the cause of the bug?
4. **What-if** I change the code like that...?

# The Four Questions to Answer

1. **What** is the bug?
2. **Why** does the bug happen? What is the cause of the bug? What is the **root cause**?
3. **Where** in the code is the cause of the bug?
4. **What-if** I change the code like that...?

> "Hercule Poirot's methods are his own. Order and method, and 'the little grey cells'." - The Big Four, in Agatha Christie

`is_prime`

# Find the Failure - Test Phase

## And Say "Eureka!"

# `is_prime` - Test

```
for number in range(10):
    print(number, is_prime(number))
```

# `is_prime` - Test

```
for number in range(10):
    print(number, is_prime(number))
```

```
0 False
1 False
2 True
3 True
4 True
5 True
6 False
7 True
8 False
9 False
```

# `is_prime` - Test

```
for number in range(10):
    print(number, is_prime(number))
```

```
0 False
1 False
2 True
3 True
4 True
5 True
6 False
7 True
8 False
9 False
```

# The Debugging Process

1. **Reproduce** (What?)
2. **Diagnose** (Why? Where?)
3. **Fix** (What-if?)
4. Repeat until you fix the bug
5. **Reflect**

# The Debugging Process

1. ➡ **Reproduce** (What?)
2. **Diagnose** (Why? Where?)
3. **Fix** (What-if?)
4. Repeat until you fix the bug
5. **Reflect**

# The Debugging Process

1. **Reproduce** (What?)
2. ➡ **Diagnose** (Why? Where?)
3. **Fix** (What-if?)
4. Repeat until you fix the bug
5. **Reflect**

**Collect Clues**

1. The Bug Itself
    1. Input/Output
    2. Error messages & Traceback
2. Reading the Code (with a critical eye)
3. Results from the Test phase
4. Debugging Toolbox (`print`, comment in/out)

# The Debugging Process

1. **Reproduce** (What?)
2. ➡ **Diagnose** (Why? Where?)
3. **Fix** (What-if?)
4. Repeat until you fix the bug
5. **Reflect**

**Model the Cause(s) of the Bug**

1. Formulate Hypothesis
2. Manipulate / Check
3. Accept / Reject
4. Keep Records

# The Debugging Process

1. **Reproduce** (What?)
2. ➡ **Diagnose** (Why? Where?)
3. **Fix** (What-if?)
4. Repeat until you fix the bug
5. **Reflect**

**Use Binary Search**

How to catch lion in the desert?

> "Once you eliminate the impossible, whatever remains, no matter how improbable, must be the truth." - Sherlock Holmes, in Doyle Arthur Conan

# The Debugging Process

1. **Reproduce** (What?)
2. **Diagnose** (Why? Where?)
3. ➡ **Fix** (What-if?)
4. Repeat until you fix the bug
5. **Reflect**

# The Debugging Process

1. **Reproduce** (What?)
2. **Diagnose** (Why? Where?)
3. **Fix** (What-if?)
4. ➡ Repeat until you fix the bug
5. **Reflect**

# The Debugging Process

1. **Reproduce** (What?)
2. **Diagnose** (Why? Where?)
3. **Fix** (What-if?)
4. Repeat until you fix the bug
5. ➡ **Reflect**

## Debug Yourself

# The Debugging Process

1. **Reproduce** (What?)
2. **Diagnose** (Why? Where?)
3. **Fix** (What-if?)
4. Repeat until you fix the bug
5. **Reflect**

> "The most effective debugging tool is still careful thought, coupled with judiciously placed print statements." - Brian Kernighan in "Unix for Beginners" (1979)

`show_day`

# Test & Debug Phases

# Your Turn!

# The Debugging Process

1. **Reproduce** (What?)
2. **Diagnose** (Why? Where?)
3. **Fix** (What-if?)
4. Repeat until you fix the bug
5. **Reflect**

# Rubber Duck Debugging

Source: UNIshop Potsdam

# Debug Phase

# Wrap-up + Q&A

> "Debugging is like being the detective in a crime movie where you are also the murderer." - Filipe Fortes

# Programming Problem Solving Model

1. Reinterpret the Problem
2. Design a Solution
3. Code
4. Test
5. Debug
6. Evaluate & Reflect

# Q&A

## Problem Solving using Python - Week 6

## Reading Code, Debug