CSCA08 Fall 2017

Week 3 - Documentation and Design

Brian Harrington & Marzieh Ahmadzadeh

University of Toronto Scarborough

September 18 - 22, 2017

Admin Stuff

- Exercises
  - Pre-runs
  - Only defining functions
- Term Test #1
  - Saturday September 30, 9-11am
  - Details on course website
  - Worth 10% of your total grade (TT#2 will be worth 15%)

Booleans

- Boolean (`bool`) is a Type in Python
    - 2 possibilities `True` and `False`
- boolean expressions are **not** strings
    - type(True) vs type("True")

Boolean Expressions

- Familiar logical operations (e.g., '<')
- A few you may have to get used to (e.g., '!=')
- 6 common `comparison operators`
    - > greater than
    - < less than
    - >= greater than or equal to
    - <= less than or equal to
    - == equal to
    - != not equal to
    - `is` the same object as (same memory location)

Logical Operators

- We can combine boolean expressions together to form new boolean expressions
- 3 standard logical operators:
  - and
  - or
  - not
- **MUST** use brackets
  - Python can do precedence, but we want our code to be clear

COMMENTING

- Why Comment?
  - Helps other people understand your code
  - Helps you understand your code
  - What seems obvious now will be completely confusing in a week/month/year's time
- May be the most important skill you'll learn this year
- Generally very poorly done
- Poor code with good commenting = fixable, Good code with poor commenting = a disaster

Internal Commenting

- Using the hash symbol #
- Written right alongside the code
- Intended for people who will be editing the code
- Goal is to explain not just **what** the code is doing, but **why** it's doing it

Internal Commenting

- Don’t need a comment for every line
- But every line should be “covered" by a comment
- Examples!

Internal Commenting

- One of the goals of this course is to get you to think before you code
- Commenting first, coding second
    - Ensures you understand what you’re trying to do before you start
    - Helps you “break the problem down"
    - Very helpful habit when exam time comes around

External Commenting

- Documentation for people **using** your code
- Accessible via **help()** function
- AKA ‘DocString’
- Following a specific format is important

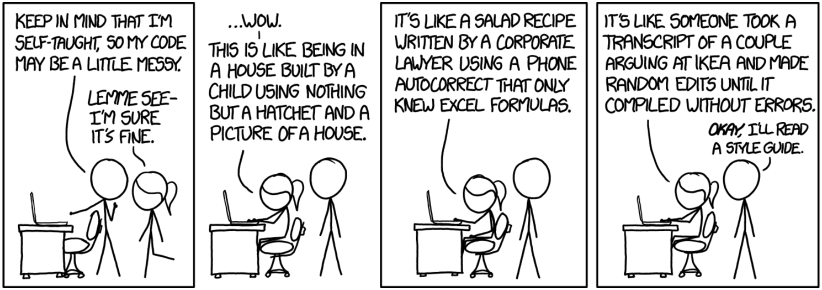
Design Recipe

- Why have a Design Recipe?
    - Make life easier for students learning to write functions
    - Enforces good commenting
    - Gives us a step-by-step process to follow when we're writing a function

Design Recipe

- 8 steps in the recipe
    1. Header: The function definition
    2. Type Contract: What types come in, what type is returned
    3. Requirements: Limits on the inputs *other* than type
    4. Examples: Some example inputs/outputs
    5. Description: Explanation of what the function does
    6. Internal Comments: Plan of what your code will do
    7. Code: The body of the function
    8. Test: Make sure that your function works

Break


KEEP IN MIND THAT I'M SELF-TAUGHT, SO MY CODE MAY BE A LITTLE MESSY.
LEMME SEE– I'M SURE IT'S FINE.
...WOW.
THIS IS LIKE BEING IN A HOUSE BUILT BY A CHILD USING NOTHING BUT A HATCHET AND A PICTURE OF A HOUSE.
IT'S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCORRECT THAT ONLY KNEW EXCEL FORMULAS.
IT'S LIKE SOMEONE TOOK A TRANSCRIPT OF A COUPLE ARGUING AT IKEA AND MADE RANDOM EDITS UNTIL IT COMPILED WITHOUT ERRORS.
OKAY, I'LL READ A STYLE GUIDE.

Step 1: Header

- Function header, using `def`
- Function name should be descriptive
- Parameter names should also be meaningful

Step 2: Type Contract

- A contract between you and whoever is using the function
- “If you give me parameters of the types I define, I will give you output of the defined type (and I won’t crash)
- If you don’t follow the contract, I’m not responsible for what happens
- Form:
    - (`parameter type list`) -> `output type`
- Inside function, in triple quotes

STEP 3: REQUIREMENTS

- Any other requirements that the user must obey
- Similar to type contract, but for things other than type
- Sometimes referred to a preconditions
- Ex: $REQ : value1 > 0$ - Tells the user that $value1$ must be positive
- Goes below type contract

Step 4: Examples

- Write a few example calls to your function, including expected output
- Standard cases
  - Don't worry about tricky/border cases yet
- go after REQs
- This step is not necessary for functions which use `random` or `i/o` (cases where couldn't predict output given input)

STEP 5: DESCRIPTION

- A description of what the function does
- Should be understandable to anyone reading it
  - They shouldn't have to know anything about the internals of your function
- Should mention every parameter by name
- Goes after type contract, but before REQs

Step 6: Internal Comments

- This is where you plan what your function will do
- Skipping this step = recipe for disaster
- Use indentation of comments to plan code
- Look at examples to help you

Step 7: Code

- This is the bit that actually gets executed
- Goal of the design recipe is that this step should be *trivial*
- Don't want to be focusing on algorithm and implementation details at the same time

Step 8: Test the Function

- Run all of your example cases from step 1
- Come up with a **Testing Plan**
- Create a `testing file`
- Run all tests
- If there are any problems, go back to earlier steps and repeat until you pass all tests

DESIGN RECIPE

- Review: 8 steps in the recipe
  1. Header: The function definition
  2. Type Contract: What types come in, what type is returned
  3. Requirements: Limits on the inputs *other* than type
  4. Examples: Some example inputs/outputs
  5. Description: Explanation of what the function does
  6. Internal Comments: Plan of what your code will do
  7. Code: The body of the function
  8. Test: Make sure that your function works

DESIGN RECIPE

- How can we remember all the steps?
  1. **H**eader
  2. **T**ype **C**ontract
  3. **R**equirements
  4. **E**xamples
  5. **D**escription
  6. **I**nternal **C**omments
  7. **C**ode
  8. **T**esting

Design Recipe

- How can we remember all the steps?
  1. **H**
  2. **T/C**
  3. **R**
  4. **E**
  5. **D**
  6. **I/C**
  7. **C**
  8. **T**

Design Recipe

- How can we remember all the steps?
  1. **H**elpful
  2. **C**omputers
  3. **R**eally
  4. **E**ven
  5. **D**umb ones
  6. **C**an
  7. **C**omplete
  8. **T**asks

Design Recipe

- How can we remember all the steps?
    1. **H**arrington
    2. **T**eaches
    3. **R**eally
    4. **E**xcellent
    5. **D**isciples
    6. **I**n
    7. **C**SCA08
    8. **T**his year

Design Recipe

- How can we remember all the steps?
  1. **H**elpful
  2. **T**As
  3. **R**arely
  4. **E**ver
  5. **D**evistate
  6. **I**ncoming
  7. **C**S students
  8. **T**erminally

Design Recipe

- (Clearly) we need your help...
  1. **H**
  2. **T/C**
  3. **R**
  4. **E**
  5. **D**
  6. **I/C**
  7. **C**
  8. **T**

Mnemonic Contest

- Who can come up with the best mnemonic?
- Post your suggestion to the forum
    - Keep it clean!
- I will pick some of my favourites, and create a survey
- The student who suggests the winning mnemonic will receive a prize, the likes of which you have never seen
    - (Assuming you have never seen a chocolate bar)