ECOR 1041
Computation and Programming

A Recipe for Designing Functions, Documentation Strings

Carleton University

References

- *Practical Programming*, 3rd ed.
  - Chapter 3, pp. 47 – 58:
    - This is section:
      - Designing New Functions: A Recipe

Lecture Objectives

- Present a lightweight 5-step process for developing Python functions
  - We call this the Function Design Recipe (FDR)

Learning Outcomes (Vocabulary)

- Know the meaning of these words and phrases
  - Type annotations, type contract
  - Documentation string
  - Precondition

Learning Outcomes

- Starting with a prose description of a function's requirements, use the *function design recipe* to develop a working, well-documented Python function

Learning Outcomes: Function Design Recipe: Summary of Steps

1. Examples
2. Header
3. Description
4. Body
5. Test

How Do We Design Functions?

- Programmers are rarely given mathematical formulas to convert into functions
- Typically, they are provided with problem descriptions expressed in prose and pictures
  - Descriptions may contain irrelevant or ambiguous information, or be incomplete

How Do We Design Functions?

"There are two kinds of people in this world:

1. Those who can extrapolate information from incomplete data.
2.

"

How Do We Design Functions?

- Early on, we need to determine what information a function takes and what information it returns
  - We must understand a function from a caller's perspective before we design and code the function body
- A *design recipe* guides us through the process of developing functions, from informal problem descriptions to tested, maintainable code

Practical Programming's Function Design Recipe (FDR)

- A 5-step recipe for designing, coding and testing functions
- A similar approach is at the core of many modern software development techniques used in industry to develop large-scale software systems (e.g., test-driven development, XP, Scrum, etc.)

Function Design Recipe: Outcomes

- A working function
- Documentation that helps us understand what the function does and how to use it (call it), without having to understand the function's algorithm

Function Design Recipe: By Example

- We will introduce the FDR by using it to write a function that solves this problem:
  - At an old-style movie theatre, every attendee pays $5 per ticket. Each performance costs the theatre $20, plus $0.50 per attendee.
  - What is the net income (profit) for the theatre, given the number of attendees at a performance?

Step 1: Examples

- Choose a descriptive function name
- Write some example function calls, showing the arguments and the return values
- Put the examples in a *documentation string* (docstring)

Example Step 1: Examples

- Name the function `total_profit`
  - It has one argument: the number of attendees
  - Something to think about: should the ticket price, the fixed cost ($20) and the cost per attendee be function arguments or constants?

Example Step 1: Examples

- Derive the formulae:
  - profit = income - cost
  - income = price per ticket $\times$ number of attendees
  - total cost = fixed cost per performance + cost per attendee $\times$ number of attendees
- Now use a calculator to determine the expected profits for different numbers of attendees

Example Step 1: Examples

- Now write the examples (start writing the docstring)

```
"""
>>> total_profit(5)
2.5
>>> total_profit(2)
-11.0
>>> total_profit(100)
430.0
"""
```

By convention, a docstring starts and ends with three double-quote characters.

Step 2: Header

- Choose descriptive parameter names
- Include *type annotations* (parameter types and return type) in the function header
- Format:

```
def fn_name(param1: type,...) -> return_type:
```

Example Step 2: Header

- `attendees` is a descriptive name for the parameter and `int` is the most appropriate type
- The function returns the profit, which is a `float`
- Type the function header:

```
def total_profit(attendees: int) -> float:
    """
    Examples not shown.
    """
```

The docstring must be indented (convention: 4 spaces)

More About Type Annotations

```
def fn_name(param1: type, ...) -> return_type:
```

Parameter types:
- `int`: the argument must be an `int`
- `float`: the argument can be a `float` or an `int`

- The type annotations form a *type contract*
  - The contract promises that, if the function is called with the correct types of arguments, it will return a value with the correct type

More About Type Annotations

- Python does not use the type annotations to check if the function call arguments have the specified types
- Example: if you call `total_profit` this way:

```
total_profit(5.0)
```

  Python will not display an error message telling you that the argument should be an int instead of a float

Step 3: Description

- At the top of the docstring, write a summary of what the function does, in terms of its parameters, and what it returns (all on one line, if possible).
  - Additional sentences may be needed to provide more information
- Describes what, not how
- Use *preconditions* to document any assumptions about the permitted values of arguments

Example Step 3: Description

- Edit the docstring to include a summary of what `total_profit` does/returns, and list the precondition (the number of attendees must be non-negative)
- The function definition now looks like this (next slide)

Example Step 3: Description

```
def total_profit(attendees: int) -> float:
    """Return the total profit of a movie performance,
    given the number of attendees.

    Precondition: attendees >= 0.

    >>> total_profit(5)
    2.5
    >>> total_profit(2)
    -11.0
    >>> total_profit(100)
    430.0
    """
```

More About Docstrings

- A *character string* is a sequence of characters
- In Python, a literal string is enclosed in a pair of single quotes, double quotes, triple single quotes or triple double quotes
- By convention, a docstring always begins and ends with triple double quotes

More About Docstrings

- The summary line is a phrase that describes what the function does as a command ("Return the total profit…"), not as a description (Returns the total profit…")
- See PEP 257, *Docstring Conventions*, for more information
  - https://peps.python.org/pep-0257/
- Python ignores docstrings – they are for humans to read

Step 4: Body

- Design the function's algorithm, write the body
- If you decide that the body should call functions that you have not yet developed, use the FDR to produce those functions

Example Step 4: Body

```
PRICE_PER_TICKET = 5
FIXED_COST = 20
COST_PER_ATTENDEE = 0.50

def total_profit(attendees: int) -> float:
    """Return the total profit of a movie performance,
    given the number of attendees.
    Rest of docstring not shown.
    """
    return PRICE_PER_TICKET * attendees \
           - FIXED_COST + attendees * COST_PER_ATTENDEE
```

YES, THERE IS A BUG!

Step 5: Test

- Use (at a minimum) the docstring examples, copied/pasted into the shell
- Consider coding a program to automate testing
  - Python provides `doctest` and `unittest` modules
  - 3rd-party testing frameworks; e.g., pytest
    - https://docs.pytest.org/en/latest/
  - More on automated testing later…

Example Step 5: Test

- Check if the actual results match the expected results (from Step 1)

```
>>> total_profit(5)
7.5
>>> total_profit(2)
-9.0
>>> total_profit(100)
530.0
```

Not the results we expected! 😢 🤬

Example Step 5: Test

- Locate the bug (we added the incremental cost for the attendees to the income instead of subtracting it)
- Edit the function definition and retest (see next slide)

Example Step 4 Revisited: Body

```
PRICE_PER_TICKET = 5
FIXED_COST = 20
COST_PER_ATTENDEE = 0.50

def total_profit(attendees: int) -> float:
    """Return the total profit of a movie performance,
    given the number of attendees.
    Rest of docstring not shown.
    """
    return PRICE_PER_TICKET * attendees \
           - (FIXED_COST + attendees * COST_PER_ATTENDEE)
```

Example Step 5 Revisited: Test

- Check if the actual results match the expected results

```
>>> total_profit(5)
2.5
>>> total_profit(2)
-11.0
>>> total_profit(100)
430.0
```

Actual results match the expected results 😀

Example Step 5 Revisited: Test

- A few docstring examples typically will not cover all cases – are additional tests required?
- Test any *edge cases*
  - 0 is at the edge between valid and invalid arguments (see the precondition), so test `total_profit(0)`

Function Design Recipe: Summary

1. Examples
2. Header
3. Description
4. Body
5. Test

Recap of Learning Outcomes

Carleton University

Learning Outcomes (Vocabulary)

- Know the meaning of these words and phrases
  - Type annotations, type contract
  - Documentation string
  - Precondition

Learning Outcomes

- Starting with a prose description of a function's requirements, use the *function design recipe* to develop a working, well-documented Python function

Learning Outcomes: Function Design Recipe: Summary of Steps

1. Examples
2. Header
3. Description
4. Body
5. Test