CPSC 103 - Introduction to Systematic Program Design

Module 02. How to Design Functions

Prof. Karina Mochetti

2020.W2

Announcements

What is new!

http://www.cs.ubc.ca/~mochetti/CPSC103.html

Goals

- Use the How to Design Functions (HtDF) recipe to design functions that operate on primitive data.
- Read a complete function design and identify its different elements.
- evaluate the elements of a function design for clarity, simplicity and consistency with each other.
- Evaluate an entire design for how well it solves the given problem.
- Explain the intended purpose of the HtDF recipe's steps in a way that generalizes to other design problems.

Quiz #02 Feedback

Good job, everyone!

We will talk about the `is_tall()` function later!

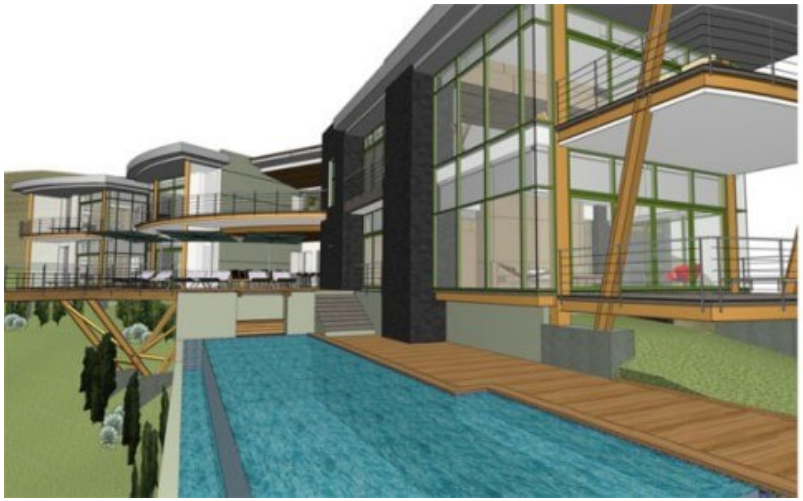
- How important will the whole process of defining function be when it comes to grading?
  You will be graded primarily on following our design process for the remainder of the term (The final correctness of your code will usually be part of the grade only).

Design

- We have a process to make clearer what a problem means.
- You know just how we want to go about solving a problem.
- The design process gives you context to understand and use the Python you learned.
- **You will be graded primarily on following our design process for the remainder of the term!**

Design

- Programming is like constructiong a build. Can you construct without a plan?
- First we need to understand the problem in different ways, then build a simplified version and finally work on the full version.



How to Design Functions recipe

The HtDF recipe consists of the following steps:

1. Signature, purpose and stub
2. Examples
3. Template
4. Code the function body
5. Test and debug until correct

Signature

Signature

It defines the input type(s) and output type for your function. It is also when you give a name to your function and to the parameters. Names should be all lower cases, with underline separating words.

Example: Design a program that returns the double of a given number.

```
@typecheck
def double(n: float) -> float:
```

Purpose

Purpose

The purpose statement describes what the function will return. It is written in triple quotes at the top of a function's body that documents the function.

Example: Design a program that returns the double of a given number.

```
@typecheck
def double(n: float) -> float:
    """
    returns n doubled
    """
```

Stub

The stub must return a value of the right type. It doesn't need to be correct, any value from the right type will do.

Example: Design a program that returns the double of a given number.

```
@typecheck
def double(n: float) -> float:
    """
    returns n doubled
    """

    return 0        # stub
```

Examples

Write at least one example of a call to the function and the result that the function is expected to return.

```
@typecheck
def double(n: float) -> float:
    """
    returns n doubled
    """

    return 0        # stub

start_testing()
expect(double(0), 2 * 0)
expect(double(-2.1), 2 * -2.1)
expect(double(10.11), 2 * 10.11)
summary()
```

Template

Comment out the body of the stub and copy the body of template from the data definition to the function design. In case there is no data definition, just copy all parameters.

```
@typecheck
def double(n: float) -> float:
    """
    returns n doubled
    """

    # return 0        # stub
    return ...(n)     # template

start_testing()
expect(double(0), 2 * 0)
expect(double(-2.1), 2 * -2.1)
expect(double(10.11), 2 * 10.11)
summary()
```

Code

Complete the function body by filling in the template, note that you have a lot of information written already. Remember to comment the template.

```
@typecheck
def double(n: float) -> float:
    """
    returns n doubled
    """

    # return 0        # stub
    # return ...(n)   # template
    return 2*n

start_testing()
expect(double(0), 2 * 0)
expect(double(-2.1), 2 * -2.1)
expect(double(10.11), 2 * 10.11)
summary()
```

Test

You should run your functions until all tests pass.

```
@typecheck
def double(n: float) -> float:
    """
    returns n doubled
    """

    # return 0        # stub
    # return ...(n)   # template
    return 2*n

start_testing()
expect(double(0), 2 * 0)
expect(double(-2.1), 2 * -2.1)
expect(double(10.11), 2 * 10.11)
summary()
```

3 of 3 tests passed

is_tall

Jupyter Code: is_tall

Open the file Module 02.ipynb.

Multiply by 4

Jupyter Code: Multiply by 4

Open the file **Module 02.ipynb**.

Questions 1 - 7

Questions?

Ask CPSC 103

http://www.cs.ubc.ca/~mochetti/askCPSC103.html

Tracing a Function: Discount

Jupyter Code: Discount

Open the file **Module 02.ipynb**.

- Computers ignore each empty line, comment line and purpose line.
- Each function has its own space in memory with its own variables.
- Computers run the code from top to bottom.
- The execution always start in the first line that is not a function definition (in our design the tests).
- When a function is called, the parameters are copied to the function memory in the order they were used on the calling.

Tracing a Function: Discount

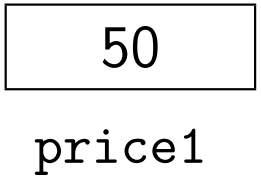
```
def discount(price1: float, price2: float)
                                        -> float:
    if price1 > price2:
        return price1*(0.95) + price2*(0.9)
    else:
        return price1*(0.9) + price2*(0.95)
```

```
price1 = 50
price2 = 75
discount(price1, price2)
```

Tracing a Function: Discount

```
def discount(price1: float, price2: float)
                                    -> float:
    if price1 > price2:
        return price1*(0.95) + price2*(0.9)
    else:
        return price1*(0.9) + price2*(0.95)
```

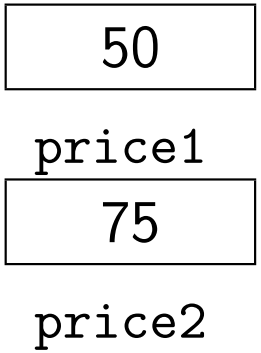
```
price1 = 50
price2 = 75
discount(price1, price2)
```



Tracing a Function: Discount

```
def discount(price1: float, price2: float)
                                        -> float:
    if price1 > price2:
        return price1*(0.95) + price2*(0.9)
    else:
        return price1*(0.9) + price2*(0.95)
```

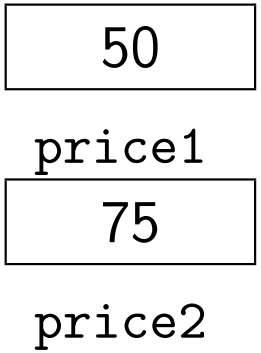
```
price1 = 50
price2 = 75
discount(price1, price2)
```



Tracing a Function: Discount

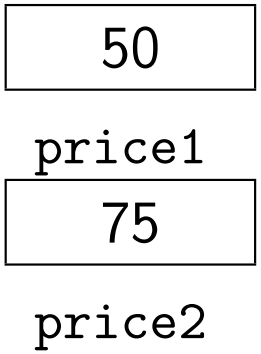
```
def discount(price1: float, price2: float)
                                        -> float:
    if price1 > price2:
        return price1*(0.95) + price2*(0.9)
    else:
        return price1*(0.9) + price2*(0.95)
```

```
price1 = 50
price2 = 75
discount(price1, price2)
```

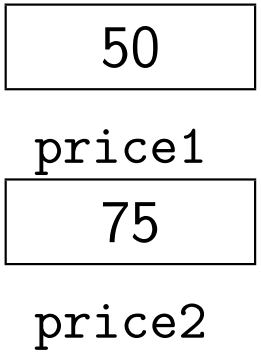


Tracing a Function: Discount

```
def discount(price1: float, price2: float)
                                    -> float:
    if price1 > price2:
        return price1*(0.95) + price2*(0.9)
    else:
        return price1*(0.9) + price2*(0.95)
```

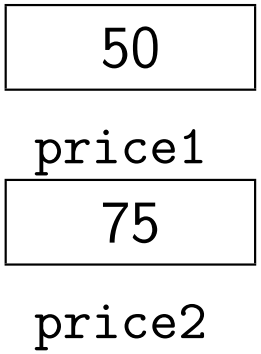


```
price1 = 50
price2 = 75
discount(price1, price2)
```

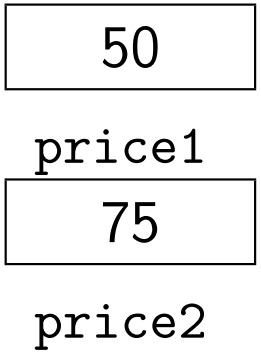


Tracing a Function: Discount

```
def discount(price1: float, price2: float)
                                    -> float:
    if price1 > price2:
        return price1*(0.95) + price2*(0.9)
    else:
        return price1*(0.9) + price2*(0.95)
```

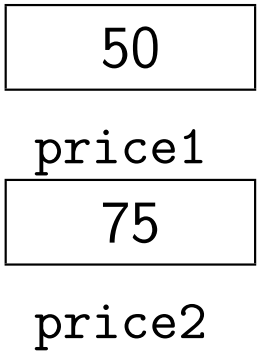


```
price1 = 50
price2 = 75
discount(price1, price2)
```

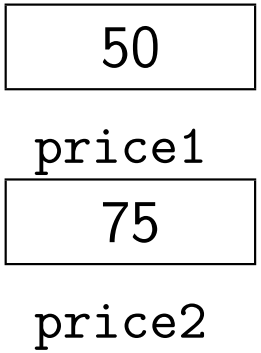


Tracing a Function: Discount

```
def discount(price1: float, price2: float)
                                    -> float:
    if price1 > price2:
        return price1*(0.95) + price2*(0.9)
    else:
        return price1*(0.9) + price2*(0.95)
```



```
price1 = 50
price2 = 75
discount(price1, price2)
```



Tracing a Function: Discount

```
def discount(price1: float, price2: float)
                                    -> float:
    if price1 > price2:
        return price1*(0.95) + price2*(0.9)
    else:
        return price1*(0.9) + price2*(0.95)
```

50

price1

75

price2

116.25

return

```
price1 = 50
price2 = 75
discount(price1, price2)
```

50

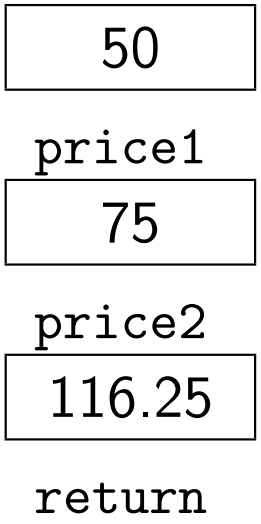
price1

75

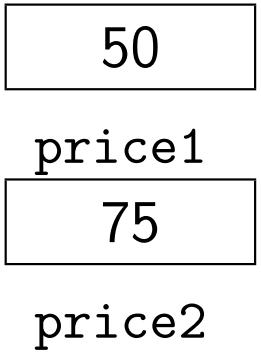
price2

Tracing a Function: Discount

```
def discount(price1: float, price2: float)
                                    -> float:
    if price1 > price2:
        return price1*(0.95) + price2*(0.9)
    else:
        return price1*(0.9) + price2*(0.95)
```



```
price1 = 50
price2 = 75
discount(price1, price2)
```



Tracing a Function: Discount

```
def discount(price1: float, price2: float)
                                    -> float:
    if price1 > price2:
        return price1*(0.95) + price2*(0.9)
    else:
        return price1*(0.9) + price2*(0.95)
```

50

price1

75

price2

116.25

return

```
price1 = 50
price2 = 75
discount(price1, price2)
```

50

price1

75

price2

Cell output on Jupyter: 116.25

Tracing a Function: Discount

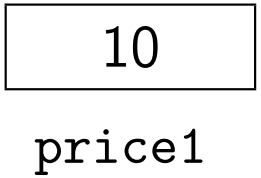
```
def discount(price1: float, price2: float)
                                    -> float:
    if price1 > price2:
        return price1*(0.95) + price2*(0.9)
    else:
        return price1*(0.9) + price2*(0.95)
```

```
price1 = 10
price2 = 20
discount(price2, price1)
```

Tracing a Function: Discount

```
def discount(price1: float, price2: float)
                                      -> float:
    if price1 > price2:
        return price1*(0.95) + price2*(0.9)
    else:
        return price1*(0.9) + price2*(0.95)
```

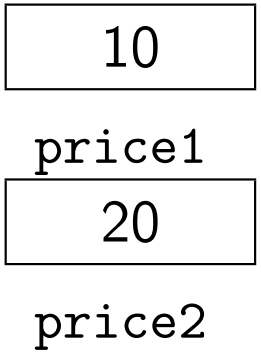
```
price1 = 10
price2 = 20
discount(price2, price1)
```



Tracing a Function: Discount

```
def discount(price1: float, price2: float)
                                        -> float:
    if price1 > price2:
        return price1*(0.95) + price2*(0.9)
    else:
        return price1*(0.9) + price2*(0.95)
```

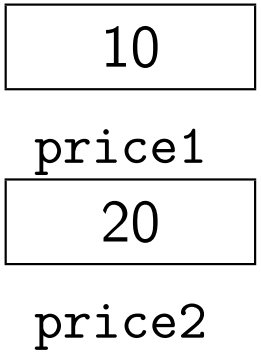
```
price1 = 10
price2 = 20
discount(price2, price1)
```



Tracing a Function: Discount

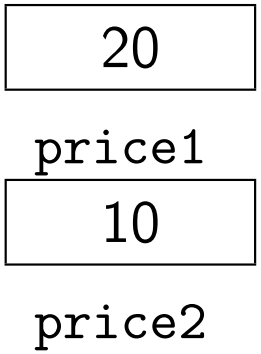
```
def discount(price1: float, price2: float)
                                        -> float:
    if price1 > price2:
        return price1*(0.95) + price2*(0.9)
    else:
        return price1*(0.9) + price2*(0.95)
```

```
price1 = 10
price2 = 20
discount(price2, price1)
```

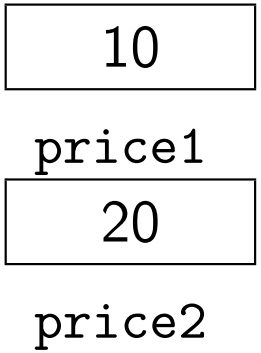


Tracing a Function: Discount

```
def discount(price1: float, price2: float)
                                    -> float:
    if price1 > price2:
        return price1*(0.95) + price2*(0.9)
    else:
        return price1*(0.9) + price2*(0.95)
```

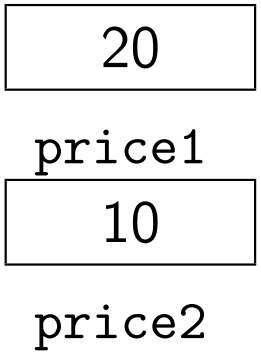


```
price1 = 10
price2 = 20
discount(price2, price1)
```

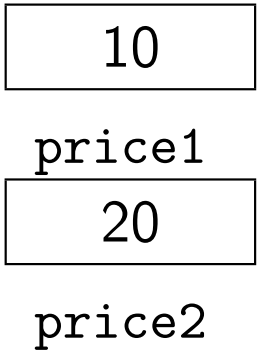


Tracing a Function: Discount

```
def discount(price1: float, price2: float)
                                    -> float:
    if price1 > price2:
        return price1*(0.95) + price2*(0.9)
    else:
        return price1*(0.9) + price2*(0.95)
```

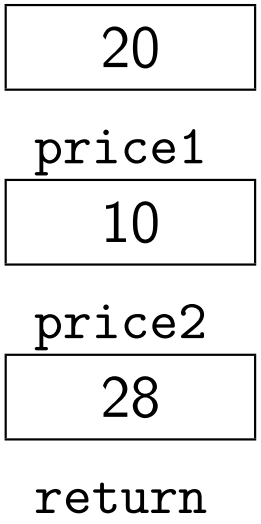


```
price1 = 10
price2 = 20
discount(price2, price1)
```

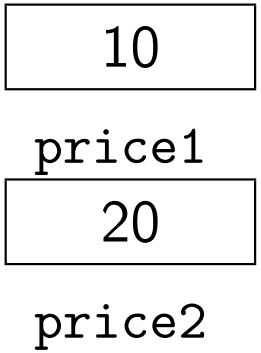


Tracing a Function: Discount

```
def discount(price1: float, price2: float)
                                    -> float:
    if price1 > price2:
        return price1*(0.95) + price2*(0.9)
    else:
        return price1*(0.9) + price2*(0.95)
```

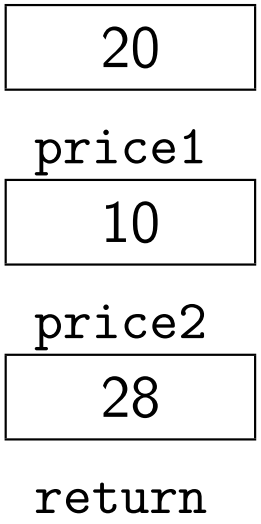


```
price1 = 10
price2 = 20
discount(price2, price1)
```

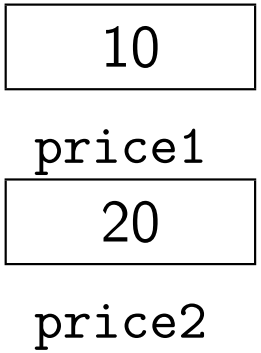


Tracing a Function: Discount

```
def discount(price1: float, price2: float)
                                    -> float:
    if price1 > price2:
        return price1*(0.95) + price2*(0.9)
    else:
        return price1*(0.9) + price2*(0.95)
```

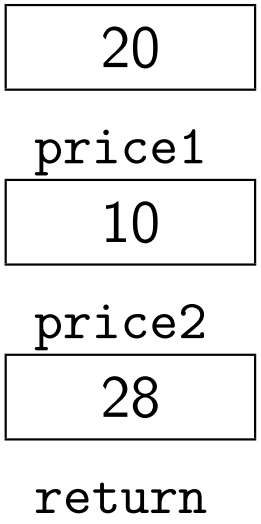


```
price1 = 10
price2 = 20
discount(price2, price1)
```

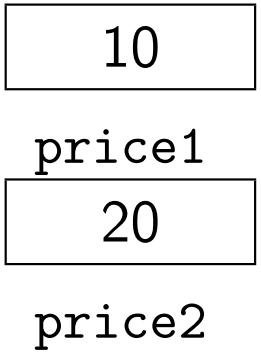


Tracing a Function: Discount

```
def discount(price1: float, price2: float)
                                    -> float:
    if price1 > price2:
        return price1*(0.95) + price2*(0.9)
    else:
        return price1*(0.9) + price2*(0.95)
```



```
price1 = 10
price2 = 20
discount(price2, price1)
```



Cell output on Jupyter: 28

Tracing a Function

Python Tutor

Tracing on paper is going to give you a MUCH better sense of what's going on! However, the Python Tutor can complete traces for you.

http://pythontutor.com/visualize.html

Caution: The Python Tutor cannot use the cs103 library. So, images, @typecheck, start_testing, summary, and expect (along with a few other things) will not work.

Jupyter Code: CPSC 103 d-tective

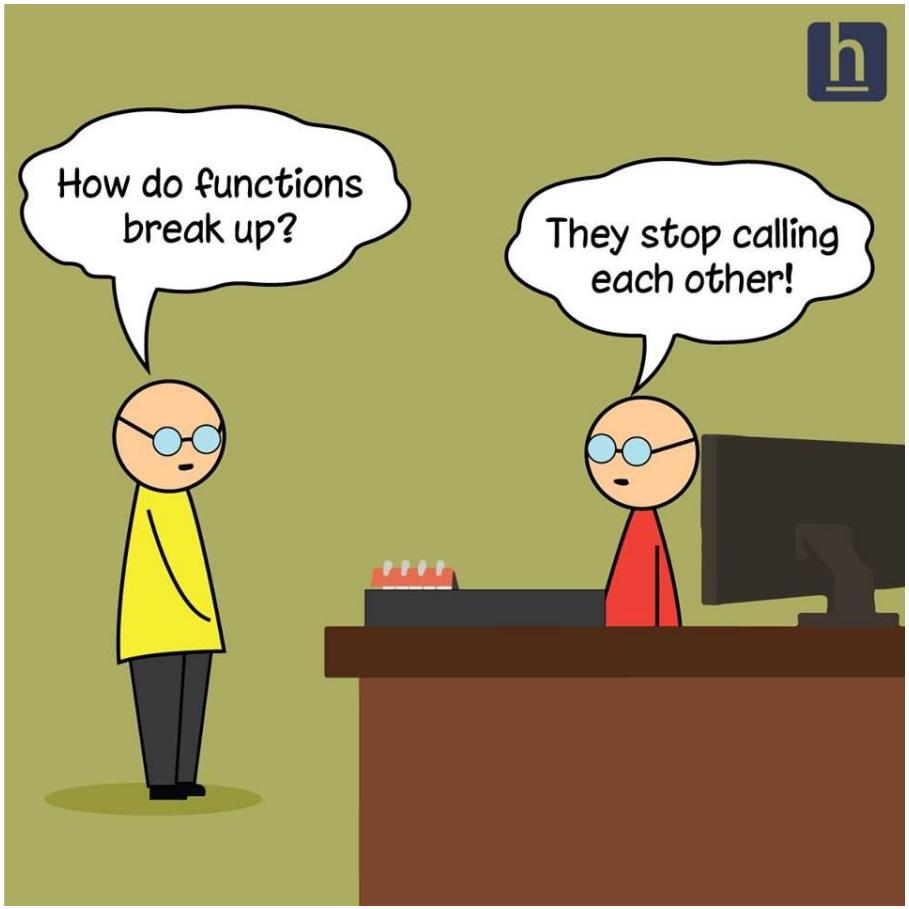
Open the file **Module 02.ipynb**.

Questions 8 - 12

End of Module

Ask CPSC 103

http://www.cs.ubc.ca/~mochetti/askCPSC103.html



Exercise

The following program calculate the final value of a product after adding a 12% tax. Find the 3 desgin errors of this program.

```
@typecheck
def AddTax(value: float) -> float:
    """
    returns the final price adding the tax of 12% on value
    """

    return 0        # stub
    #return value   # template
    return value*0.12 + value

start_testing()
expect(AddTax(10), 11.2)
expect(AddTax(12), 13.44)
expect(AddTax(500), 560)
summary()
```

Exercise: Solution

1. Forget to comment stub

```
@typecheck
def AddTax(value: float) -> float:
    """
    returns the final price adding the tax of 12% on value
    """

    #return 0         # stub
    #return value   # template
    return value*0.12 + value

start_testing()
expect(AddTax(10), 11.2)
expect(AddTax(12), 13.44)
expect(AddTax(500), 560)
summary()
```

Exercise: Solution

2. Function name is not written in lowercases

```
@typecheck
def add_tax(value: float) -> float:
    """
    returns the final price adding the tax of 12% on value
    """

    #return 0        # stub
    #return value  # template
    return value*0.12 + value

start_testing()
expect(add_tax(10), 11.2)
expect(add_tax(12), 13.44)
expect(add_tax(500), 560)
summary()
```

Exercise: Solution

3. Template is incorrect.

```
@typecheck
def add_tax(value: float) -> float:
    """
    returns the final price adding the tax of 12% on value
    """

    #return 0        # stub
    #return ...(value)  # template
    return value*0.12 + value

start_testing()
expect(add_tax(10), 11.2)
expect(add_tax(12), 13.44)
expect(add_tax(500), 560)
summary()
```