Today’s Plan

• Docstrings
• More on modules and the meaning of `if __name__ == "__main__":`
• How to automatically test your docstring examples
  • The magic of doctest!! 😊
• Likely just start talking about Boolean type, relational operators, and if-statements.
Docstrings

• A docstring (Documentation String) should:
  • Immediately follow the function header
  • Precede all other statements
  • Be enclosed in triple quotes: "" ... "" or """" ... """
• Include:
  • The type contract
  • A description of the function
  • And examples.
A Function

```python
def triangle_area(triangle_base, triangle_height):
    """
    (number, number) -> float
    
    Precondition: triangle_base > 0 and triangle_height > 0
    
    Return the area of a triangle with base triangle_base and height
    triangle_height.
    
    >>> triangle_area(100, 2)
    100.0
    >>> triangle_area(12.5, 34.0)
    212.5
    """
    return triangle_base * triangle_height / 2
```

Function Header

Function’s Docstring

Function Body
def triangle_area(triangle_base, triangle_height):
    """
    (number, number) -> float
    Precondition: triangle_base > 0 and triangle_height > 0
    Return the area of a triangle with base triangle_base and height triangle_height.
    """
    return triangle_base * triangle_height / 2

>>> triangle_area(100, 2)
100.0
>>> triangle_area(12.5, 34.0)
212.5
Type Contract Format

(param1_type, param2_type, ..., last_param_type) -> ret_value_type

For example, a function with two parameters of type int that returns a value of type float should have the following type contract:

(int, int) -> float
Writing Good Docstrings: The Description

• The function description should start with the word Return (write it as a command).
  • Except if the function returns value None. We’ll do more examples about this later on.

• You should mention all function parameters by name in your description.

• If there are some conditions that your parameters should respect, beyond what was specified in the type contract, write a Precondition. This is placed a line before the function description, and is clearly marked. For example: Precondition: x >= 0

• Your description should not specify how the function does what it’s supposed to. Just what it does.

• You should not mention the types of the parameters/return values in your description. You have already done so in your type contract, earlier in the docstring.
Docstring Examples

• Two examples is usually a good number.
  • But you should test your function with more!! Especially for corner cases.

• The arguments used in your function call examples should also respect the type contract.
  • If a function has a parameter int, you should not call it with a string!

• Same for the return value displayed.
  • If your function always returns a float, your example should show that as well.
A last note on docstring formatting

• Insert a blank line between
  • the type contract and the description
  • the description and the docstring examples

• If there is a precondition, add a new line before and after it as well.
Update

• On Sept. 21\textsuperscript{st}, I corrected a typo in the description of function add in the grades.py and grades_program.py posted under Lectures.

• I had used num1 and num2 instead of number1 and number2. If you recall, we changed the parameter names last minute in class.

• The updated files have an annotation at the very bottom, clearly stating the correction.
What is a module?

• Module Definition (from your textbook):
  • “A module is a collection of variables and functions that are grouped together in a single file.”
  • “The variables and functions in a module are usually related to one another in some way; for example, module math contains the variable pi and mathematical functions [...]”

• Importing a module (using import module_name) gives us access to its functions, variables, etc.

• Every module has a special variable called __name__
Modules (cont’d)

• You can run modules **directly** or **indirectly**.
  • In Wing, you run a module directly when you press the green triangle.
  • You run a module indirectly when you import it.

• What is the use of the `__name__` variable?
  • The module you are running **directly** will have its special `__name__` variable refer to “*main*”
  • Any module you are running **indirectly** (that you have imported) will have its `__name__` variable refer to its `filename` (without the `.py` extension).
Exercise 1

• Assume a module called test1_module.py exists, that has the following line of code: print(__name__)

What will be printed if I run test1_module directly?

___main___

What will be printed if I import test1_module?

test1_module
Modules (cont’d)

• If there is some code you only want to run if and only if a module is run directly, you enclose that code in the following

```python
if __name__ == "__main__":
    # Anything you put here will be executed
    # only if this module is run directly.
```

• Doing so ensures this code will NOT execute if you are importing this module.

• Only one such if statement is needed per file.
Exercise 2

• Assume a module called test1_module_v2.py exists, that has the following lines of code:
  if __name__ == "__main__":
    print(__name__)

• Also assume a module named test2.py exists in the same directory with the following lines of code:
  import test1_module_v2
  print("In test2.py __name__ is", __name__)

• What will be printed if you run test2.py directly?
  In test2.py __name__ is __main__