

COMPSCI 101 Principles of Programming

Lecture 28 – Docstrings, Doctests



- At the end of this lecture, students should be able to
 - use Doctest by including simple tests in function docstrings

2



docstring

- A docstring is a special kind of string used to provide documentation
 - Appears at the top of every program
 - three double-quotes are used to surround the docstring
 - All programs should include a docstring at the beginning of the program
 - ▶ The docstring contains the author and usually a version number
 - As well as the docstring describing the purpose of the program, amost important recommendation is the common sense: be short, clear and concise! def get_the_fib(which_fib):

```
docstring

def main():
   hours = 5
   minutes = 23
   total_minutes = hours * 60 + minutes
   print(total_minutes)

main()
```



Errors

- No matter how smart or how careful you are, errors are your constant companion.
- With practice, you will get better at not making errors, and much, much better at finding and correcting them.
- ▶ There are three kinds of errors:
 - syntax errors,
 - runtime errors, and
 - logic errors.

i



Syntax Errors

- ▶ These are errors where Python finds something wrong with your program, and you can't execute it.
 - ▶ mostly typos missing punctuation , wrong indentation, case sensitive ...
- Syntax errors are the easiest to find and correct. The compiler will tell you where it got into trouble. Usually the error is on the exact line indicated by the compiler, or on the line just before it;

```
def main():
    number = 4
    print(number)
    for i in range(1, print("hello")
main()

Missing colon, missing ')'

SyntaxError: invalid syntax
No output regarding the number

File "Example01.py", line 4

for i in range(1, number)

^
SyntaxError: invalid syntax
```

CompSci 101



Execution/Runtime Errors

- If there are no syntax errors, Python may detect an error while your program is running
- ▶ For example: IndexError, Division by 0 etc
- Runtime errors are moderate in difficulty. Python tells you where it discovered that your program went wrong, but you need to trace back from there to figure out where the problem originated.

```
def main():
    number = 0
    print(number)
    print(230 / number)

main()

the interpreter tries to give useful information

Output:

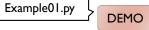
0
Traceback (most recent call last):
File "Example01.py", line 6, in <module>
    main()
File "Example01.py", line 4, in main
    print(230 / number)
ZeroDivisionError: division by zero
```

CompSci 101

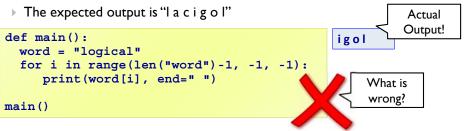


5

Logical Errors



- A <u>logical error</u>, or <u>bug</u>, is when your program compiles and runs, but does the <u>wrong</u> thing.
- ▶ The Python system, of course, has no <u>idea</u> what your program is **supposed to do**, so it provides <u>no</u> additional information to help you find the error.
- Logical errors are often difficult to find and correct.
- Example: We would like to print a string in a reverse order:



200

6

Types of errors continued

▶ Logical – harder to find, harder to correct

```
x = int(input("x: "))
y = int(input("y: "))

if x > 10:
    if y == x:
        print("Fine")

else:
    print("So what?")
```

```
x = int(input("x: "))
y = int(input("y: "))

if x > 10:
    if y == x:
        print("Fine")
    else:
        print("So what?")
```

```
x: 3
y: 3
```

- x: 3 y: 3
- Complete the output for code A and code B above?
- Which was the intention?

7

Testing is important!

Expensive Fireworks (1996)

- In 1996, code from the Ariane 4 rocket is reused in the Ariane 5, but the new rocket's faster engines trigger a bug in an arithmetic routine inside the flight computer.
- The error is in code to convert 64-bit floating-point numbers to a 16-bit signed integers. The faster engines cause the 64-bit numbers to be larger, triggering an overflow condition that crashes the flight computer.
- As a result, the rocket's primary processor overpowers the rocket's engines and causes the rocket to disintegrate only 40 seconds after launch.





The Fibonacci Sequence

- ▶ The Fibonacci Sequence is the series of numbers:
 - I, I, 2, 3, 5, 8, 13, 21, 34, ...
 - The next number is found by adding up the two numbers before it.
 - The 2 is found by adding the two numbers before it (1+1)
 - Similarly, the 3 is found by adding the two numbers before it (1+2),
 - And the 5 is (2+3),
 - and so on!
 - Here is a longer list:

x_5	x_6	x_7	x_8	x_9	x_{10}
5	8	13	21	34	55
			7		



Example 02.py

n		2	3	4	5	6	7	8	9	10	Ш	12
	1	1	2	3	5	8	13	21	34	55	89	144

10



First Attempt:

Example02.py **DEMO**

▶ Complete the following function which prints the fibonacci numbers up to but **not including** up to number:

```
def print fibs(up to number):
  prev fib = 1
  next fib = 1
  while next fib < up to number:
       print(next fib, end=" ")
       prev fib, next fib = next fib, next fib + prev fib
                                              up to 20: 1 2 3 5 8 13
print("up to 20:", end = " ")
                                              up to -4:
print fibs(20)
                                              up to 0:
print("up to -4:", end = " ")
                                              up to 1:
print fibs(-4)
                                              up to 2: 1
                                                         Missing some
                In order to test the correctness of the
                                                            values
               function, we need to check with different
```

parameters (valid and invalid values)



12

🚅 Example 3

▶ Returns a list of the required number (given by how many) of fibonacci numbers:

```
def get fibs list(how many):
def get fibs list(how many):
  prev fib = 0
  next fib = 1
  fib list = []
  while len(fib list) < how many:
       prev fib, next fib = next fib, next fib + prev fib
  return fib list
print("List of first 5 fib numbers:", get fibs list(5))
```



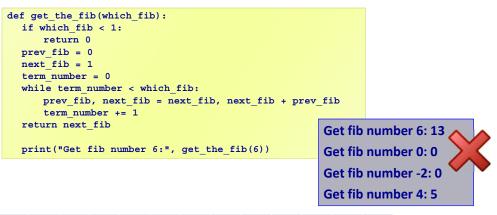
DEMO

List of first 5 fib numbers [1, 1, 2, 3, 5] List of first 0 fib numbers: [] List of first -2 fib numbers: []



Example02.py DEMO

▶ Returns the nth (given by which_fib) fibonacci number:



n	1	2	3	4	5	6	7	8	9	10	Ш	12
	1	I	2	3	5	8	13	21	34	55	89	144

13

333

Using the interactive interpreter

Note: The interactive interpreter can be used to check and run Python code interactively.

14



🜌 Remember – using docstrings

- We used docstrings to state the purpose of the program and to print the module author.
 - ▶ This is the program documentation.
 - Remember: be short, clear and concise! Other programmers, who use/improve your module, will be using your docstring as documentation.
 - Docstrings can also be added to our functions. A docstring containing the purpose of the function should be added to the docstring.

```
def get_the_fib(which_fib):
    """Returns the nth (given by which_fib) Fibonacci number.
    """
    prev_fib = 0
    next_fib = 1
    ...
```



Testing using doctest module

Example03.py

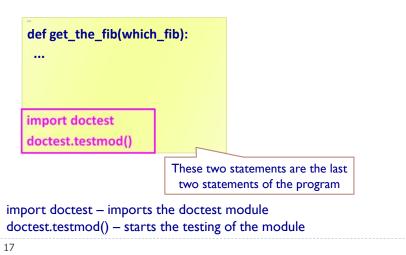
DEMO

▶ Put all your <u>test cases</u> into your docstrings

```
def cube(x):
                            File "Example02.py", line 7, in main .cube
       returns ...
                            Failed example:
       >>> cube(0)
                             cube(2)
                            Expected:
       >>> cube(1)
                                     Test Failed
 Test
                            Got:
       >>> cube(2)
cases
       >>> cube(10)
                            File "Example02.py", line 9, in main .cube
       1000
                            Failed example:
                             cube(10)
        return x * x
                            Expected:
                             1000
   import doctest
                                      Test Failed
                            Got:
   doctest.testmod()
                            ***Test Failed*** 2 failures.
16
```



If we want to include doctests in functions, we need to include the following two statements at the end of our code:



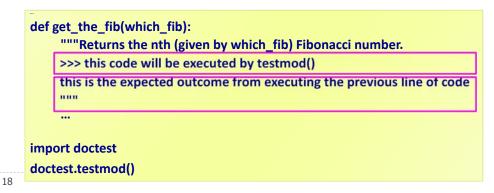
Running a program which contains doctests

- Note that in the program a main() function can be included or it can be **left out** if you just wish to just run the doctests.
- When you run the doctests (e.g., run the program on the previous slide), there is no output if the tests cause no problem, i.e., if the outcome of the tests is exactly the same as the outcome stated.
- If the outcome of the test is different, then the test **fails** and the doctest gives useful information.



Doctests –testmod() does the testing

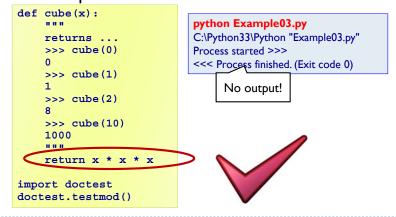
- ▶ A docstring can also contain testing code.
- Any code in our function docstrings which looks like interactive code,
 - i.e., any line in the docstring which starts with the interactive interpreter prompt, ">>>" will be executed and the outcome of the code will be compared with the stated expected outcome.





Testing using the doctest module

- ▶ Put all your <u>test cases</u> right into your doc strings
- ▶ When this program is run, there is no output because all the doctests pass.



19



Run your program using -v option, and doctest prints a detailed log of what it's trying, and prints a summary at the end:

```
def cube(x):
    """
    returns ...
    >>> cube(0)
    0
    >>> cube(1)
    1
    >>> cube(2)
    8
    >>> cube(10)
    1000
    """
    return x * x * x
```

import doctest

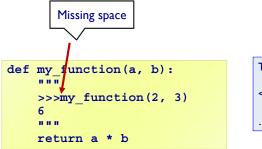
doctest.testmod()

```
python Example02.py -v
...
Trying:
    cube(2)
Expecting:
    8
    ok
Trying:
    cube(10)
Expecting:
    1000
    ok
...
4 passed and 0 failed.
Test passed.
```

21 CompSci 101



▶ No blank space after the '>>>' prompt sign:



Traceback (most recent call last):
File "Example03.py", line 12, in
<module>
doctest.testmod()
...

Example05.py

DEMO

22 CompSci 101

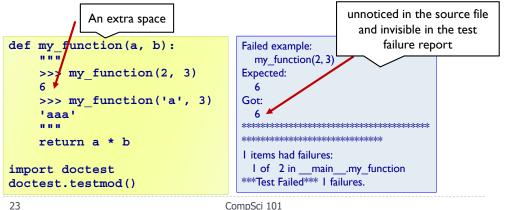


Common Problem 2

Example06.py DEMO

If the outcome doesn't match exactly (including trailing spaces), the test fails, e.g.,

▶ Example: embedded whitespace can also cause tricky problems with tests. This example has a single extra space after the 6.





Common Problem 3

Example07.py DEMO

No blank line after the expected outcome − in this case any text on the next line is considered to be part of the output, e.g.,

```
def my function(a, b):
      >>> my function(2, 3)
                                         Failed example:
                                           my function(2, 3)
      more comment
                                         Expected:
      >>> my function('a', 3)
      'aaa
                                          more comment
                                         Got:
      return a * b
                                         *****
                                         ****
                                         I items had failures:
   Doctest considers that the line
                                          I of 2 in main .my function
   "more comment" is part of the
                                         ***Test Failed*** I failures.
   output. Therefore the test fails.
24
                                  CompSci 101
```



Blank lines are used to delimit tests.

- In real world applications, output usually includes whitespace such as blank lines, tabs, and extra spacing to make it more readable.
- Blank lines, in particular, cause issues with doctest because they are used to delimit tests.
 Process started >>>

<<< Process finished. (Exit code 0)

```
def my_function(a, b):
    """
    >>> my_function(2, 3)
    delimit tests

>>> my_function('a', 3)
    'aaa'
    """
    return a * b
...
```

25 CompSci 101



Common Problem! - 4

DEMO

Write a function which takes a list of input lines, and prints them double-spaced with <u>blank lines</u> between.

```
def double space(lines):
     """Prints a list of lines double-spaced.
    >>> double space(['Line one.', 'I Expected:
    Line one.
                                              Line one.
                                              Line two.
                        interprets the blank
    Line two.
                                            Got:
                        line after Line one. in
                                              Line one.
                        the docstring as the
                                              <BLANKLINE>
                         end of the sample
    for 1 in lines:
                                              Line two.
                              output
                                              <BLANKLINE>
         print(1)
                                            **********
         print()
                                            ********
    return
                                            I items had failures:
                                             I of I in main .double space
import doctest
                                            ***Test Failed*** | failures.
doctest.testmod()
```

CompSci 101



Solution

Using <BLANKLINE>

```
def double_space(lines):
    """Prints a list of lines double-spaced.
    >>> double_space(['Line one.', 'Line two.'])
    Line one.
    <BLANKLINE>
    Line two.
    <BLANKLINE>
        """
    for l in lines:
        print(l)
        print()
    return

import doctest
doctest.testmod()
```



28

26

doctests - exercise 1

Exercise01.py

Example08.py

```
def get the fib(which fib):
   """Returns the nth Fibonacci number.
                               Do the two doctests pass or fail?
  >>> get the fib(8)
   >>> get_the_fib(5)
   if which fib < 1:
     return 0
  prev fib = 0
  next fib = 1
   fib number = 0
  while fib number < which fib:
     prev fib, next fib = next fib, next fib + prev fib
     fib number += 1
   return next fib
import doctest
doctest.testmod()
```



doctests – exercise 2

```
def get fibs list(how many):
   """Returns a list of Fibonacci numbers.
   The parameter is the number of terms in the list.
                                   Write two (useful and different)
                               doctests for the get fibs list() function.
   prev fib = 0
   next fib = 1
   fib list = []
   while len(fib list) < how many:
       fib list.append(next fib)
       prev fib, next fib = next fib, next fib + prev fib
   return fib list
import doctest
doctest.testmod()
```

29

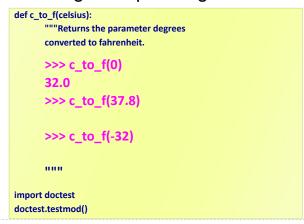
Summary

- In a Python program:
 - docstrings can be associated with modules and with functions
 - > simple tests can can be added to the docstring of a function. These tests are automatically carried out.



Converting Celsius - Fahrenheit

▶ Often, before writing the code, we know what outcomes we are expecting. These expected outcomes can be added to the function being developed using doctests.



Celsius	Fahrenheit
-35.000	-31.000
-30.000	-22.000
-25.000	-13.000
-20.000	-4.0000
-15.000	5.0000
-10.000	14.000
-5.0000	23.000
0.0000	32.000
5.0000	41.000
10.000	50.000
15.000	59.000
20.000	68.000
25.000	77.000
30.000	86.000
35.000	95.000
40.000	104.00
45.000	113.00
50.000	122.00
55.000	131.00

Exercise03.py

30