Announcements

Unable to enroll because of section conflicts? Check Student Center regularly to see if space opens up.

Install Python, Komodo Edit, and the "Run Python Module" button. The first assignment is coming next week.

No reading for next time: Our treatment of objects differs significantly from the book’s.

Slides by D. Gries, L. Lee, S. Marschner, W. White

Users Want Functions

Given: info contains a comma-separated string with last name, difficulty, execution, and penalty.

Example: info = "RAISMAN, 6.7, 9.1, 0"

Goal: store the difficulty as a string, with no extra spaces or punctuation, in variable df

Users (including other programmers) want to write things like:

raisman_df = gym.dscore("RAISMAN, 6.7, 9.1, 0")
ponor_df = gym.dscore(" PONOR, 6.2, 9.0, 0")

The function dscore is in module (file) gym.

When called, it returns a value that the user can utilize as they wish.

Parameters: Variables Holding Input Values

def dscore(info):
    """Returns: difficulty score, as a float, represented in info.
    Precondition: info is a string with commas separating its component values: last name, difficulty score, execution score, penalty."""

    startcomma = info.index(’,’)
    tail = info[startcomma+1:]
    endcomma = tail.index(’,’)
    return float(tail[endcomma:].strip())

Anatomy of a Function Definition (I)

In file gym, we define dscore as follows.

\textbf{header} \begin{itemize}
\item \textbf{def} dscore(info):
\end{itemize}

\textbf{specification} \begin{itemize}
\item \textbf{""""Returns:} difficulty score, as a float, represented in info.
\item \textbf{Precondition:} info is a string with commas separating its component values: last name, difficulty score, execution score, penalty."""
\end{itemize}

\textbf{body (indented)} \begin{itemize}
\item startcomma = info.index(’,’)
\item tail = info[startcomma+1:] # part of info after 1st ,
\item endcomma = tail.index(’,’)
\item \textbf{return} float(tail[endcomma:].strip())
\end{itemize}

Anatomy of a Specification: User Documentation

\textbf{def dscore(info)}:

\begin{itemize}
\item \textbf{""""Returns:} difficulty score, as a float, represented in info.
\item \textbf{Precondition:} info is a string with commas separating its component values: last name, difficulty score, execution score, penalty."""
\end{itemize}

\textbf{body (indented)} \begin{itemize}
\item startcomma = info.index(’,’)
\item tail = info[startcomma+1:] # part of info after 1st ,
\item endcomma = tail.index(’,’)
\item \textbf{return} float(tail[endcomma:].strip())
\end{itemize}
A Specification is a Contract

Preconditions are a promise that:
- if the arguments satisfy the preconditions, the function works as described in the specification;
- but, if the user's arguments violate the precondition, all bets are off.

>>> gym.dscore("R; 6.7, 9.0")
"I'm sorry Dave, I'm afraid I can't do that"

So write these contracts carefully!

Common sources of software errors:
- Preconditions not documented properly
- Functions used in ways that violate preconditions

Testing Program "Correctness"

• Bug: Error in a program. (Always expect them!)
• Debugging: Process of finding bugs and removing them.
• Testing: Process of analyzing, running program, looking for bugs.
• Test case: A set of input values, together with the expected output.

Get in the habit of writing test cases for a function from the function's specification — even before writing the function's body.

Testing Program "Correctness"

def number_vowels(w):
    """Returns: number of vowels in word w.
    Precondition: w string w/ at least one letter and only letters"
    pass  # nothing here yet!

Organizing Test Cases: Unit Tests

• A unit test is a module that tests another module
  • It imports the other module (so it can access it)
  • It imports the unittest module (provided by us)
  • It defines one or more test procedures
    • Evaluate the function(s) on the test cases
    • Compare the result to the expected value
  • It has special code that calls the test procedures
• The test procedures use the unittest function

Example unit test: last_name_first(n)

# test procedure
def test_last_name_first():
    """Test procedure for last_name_first(n)"
    unittest.assertEqual('White, Walker',
                         last_name_first('Walker White'))
    unittest.assertEqual('White, Walker',
                         last_name_first('Walker White '))

# Application code
if __name__ == '__main__':
    test_last_name_first()
    print 'Module name is working correctly'

Aside: Application Code

Applications often have "application code"
- Code not executed if imported; only if run as app/
  Komodo "Run Python Module" button
- Indented under the line
  if __name__ == '__main__':

Debugging with Print Statements

Print statements expose the values of variables, so you can check if they have the value you expect.

print 'in this solution, df is : ' + df + ':'

Don't leave these in your finished code! They reduce readability.