DOCUMENTATION AND TESTING

ERIC MARTIN

Part 1. Standard documentation

1. List of names in caller's scope

```
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__',
'__spec__']
>>> import math
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__',
'__spec__', 'math']
>>> from random import randrange
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__',
'__spec__', 'math', 'randrange']
>>> x = 10
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__',
'__spec__', 'math', 'randrange', 'x']
```

2. List of built-in names

```
>>> dir(__builtins__)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException',
...
'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']

>>> [name for name in dir(__builtins__) if name[0].islower()]
['abs', 'all', 'any', 'ascii', 'bin', 'bool', 'bytearray', 'bytes', 'callable', 'chr',
...
'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
```

Note: __builtins__ can be replaced by builtins if builtins has been imported.

3. Help on the built-in module

```
>>> help(__builtins__)
Help on built-in module builtins:

NAME
    builtins - Built-in functions, exceptions, and other objects.
...
FILE
    (built-in)
```

Note: __builtins__ can be replaced by 'builtins', or by builtins if builtins has been imported.

Date: Session 2, 2017.

4. Help on a built-in function

```
>>> help(__builtins__.sorted)
Help on built-in function sorted in module builtins:

sorted(iterable, /, *, key=None, reverse=False)
    Return a new list containing all items from the iterable in ascending order.

    A custom key function can be supplied to customize the sort order, and the
    reverse flag can be set to request the result in descending order.
```

Note: __builtins__.sorted can be replaced by 'builtins.sorted', or by builtins.sorted if builtins has been imported.

5. List of names in a class in the built-in module

```
>>> dir(str)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',
...
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']

>>> [name for name in dir(str) if not name.startswith('__')]
['capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs',
...

'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

Note: The name of the class, here str, can be replaced by an instance from that class, here for instance '', or 'why not?', or '__builtins__'.

6. Help on a class in the built-in module

```
>>> help(str)
Help on class str in module builtins:

class str(object)
 |  str(object='') -> str
 |  str(bytes_or_buffer[, encoding[, errors]]) -> str
...
 |  maketrans(x, y=None, z=None, /)
 |      Return a translation table usable for str.translate().
 |
 |      If there is only one argument, it must be a dictionary mapping Unicode
 |      ordinals (integers) or characters to Unicode ordinals, strings or None.
 |      Character keys will be then converted to ordinals.
 |      If there are two arguments, they must be strings of equal length, and
 |      in the resulting dictionary, each character in x will be mapped to the
 |      character at the same position in y. If there is a third argument, it
 |      must be a string, whose characters will be mapped to None in the result.
```

Note: Here, str can be replaced by '' but no other string literal because nonempty string literals are expected to be names of modules or qualified module attributes, and either the name is a valid module name or qualified module attribute in which case documentation for that module or qualified module attribute will be displayed (as demonstrated before and later), or the name is invalid and help will just report so. But taking the class list as another example, list could be replaced by any list literal, for instance [1, 2, 3], not only [].

7. Help on a method in a class in the built-in module

```
>>> help(str.upper)
Help on method_descriptor:

upper(...)
    S.upper() -> str

    Return a copy of S converted to uppercase.
```

Note: The name of the class, here str, can be replaced by an instance from that class, here for instance '', or 'why not?', or '__builtins__'.

8. List of names in a module

```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'acos',
...
'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
>>> [name for name in dir(math) if not name.startswith('__')]
['acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos',
...
'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

9. Help on a module

```
>>> help('math')
Help on module math:

NAME
    math
...
DATA
    e = 2.718281828459045
    inf = inf
    nan = nan
    pi = 3.141592653589793
    tau = 6.283185307179586

FILE
    /Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/...
```

Note: 'math' can be replaced by math if math has been imported.

10. Help on a name in a module

```
>>> help('math.log2')
Help on built-in function log2 in math:

math.log2 = log2(...)
    log2(x)

    Return the base 2 logarithm of x.
>>> import math
>>> help(math.log2)
Help on built-in function log2 in module math:

math.log2 = log2(...)
    log2(x)

    Return the base 2 logarithm of x.
>>> from math import trunc
>>> help(trunc)
Help on built-in function trunc in module math:

trunc(...)
    trunc(x:Real) -> Integral

    Truncates x to the nearest Integral toward 0. Uses the __trunc__ magic method.
```

Part 2. Documentation from modules we create

11. List of names in a module

```
>>> import rational_number
>>> dir(rational_number)
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__',
...
'gcd', 'get', 'get_both_inputs', 'output_result', 'reduce_fraction']
>>> [name for name in dir(rational_number) if not name.startswith('__')]
['compute_fraction', 'determine_reduced_fraction',
...
'gcd', 'get', 'get_both_inputs', 'output_result', 'reduce_fraction']
```

12. Help on a module thanks to doctrings

```
>>> import rational_number
>>> help(rational_number)
Help on module rational_number:

NAME
    rational_number

DESCRIPTION
    A number of functions to essentially, get as input two strings of digits,
    sigma and tau, and compute natural numbers a and b such that a / b
    is reduced and equal to 0.(sigma)(tau)(tau)(tau)(tau)...

FUNCTIONS
    compute_fraction(sigma, tau)
        Based on the computation
                0.(sigma)(tau)(tau)(tau)...
              = sigma * 10^{-|sigma|} + tau(10^{-|sigma|-|tau|} +
                                            10^{-|sigma|-2|tau|} +
                                            ...)
              = sigma * 10^{-|sigma|} +
                tau * 10^{-|sigma|-|tau|} / (1 - 10^{-|tau|})
              = sigma * 10^{-|sigma|} +
                tau * 10^{-|sigma|} / (10^{|tau|} - 1)
              = [sigma * 10^{-|sigma|} * (10^{|tau|} - 1) + tau * 10^{-|sigma|}] /
                (10^{|tau|} - 1)
              = [sigma * (10^{|tau|} - 1) + tau] /
                [(10^{|tau|} - 1) * 10^{|sigma|}]

        >>> compute_fraction('0', '0')
        (0, 90)
        >>> compute_fraction('0', '1')
        (1, 90)
        >>> compute_fraction('1', '0')
        (9, 90)
...
FILE
    /Users/emartin/Documents/COMP9021/Lectures/Lecture_2/rational_number.py
```

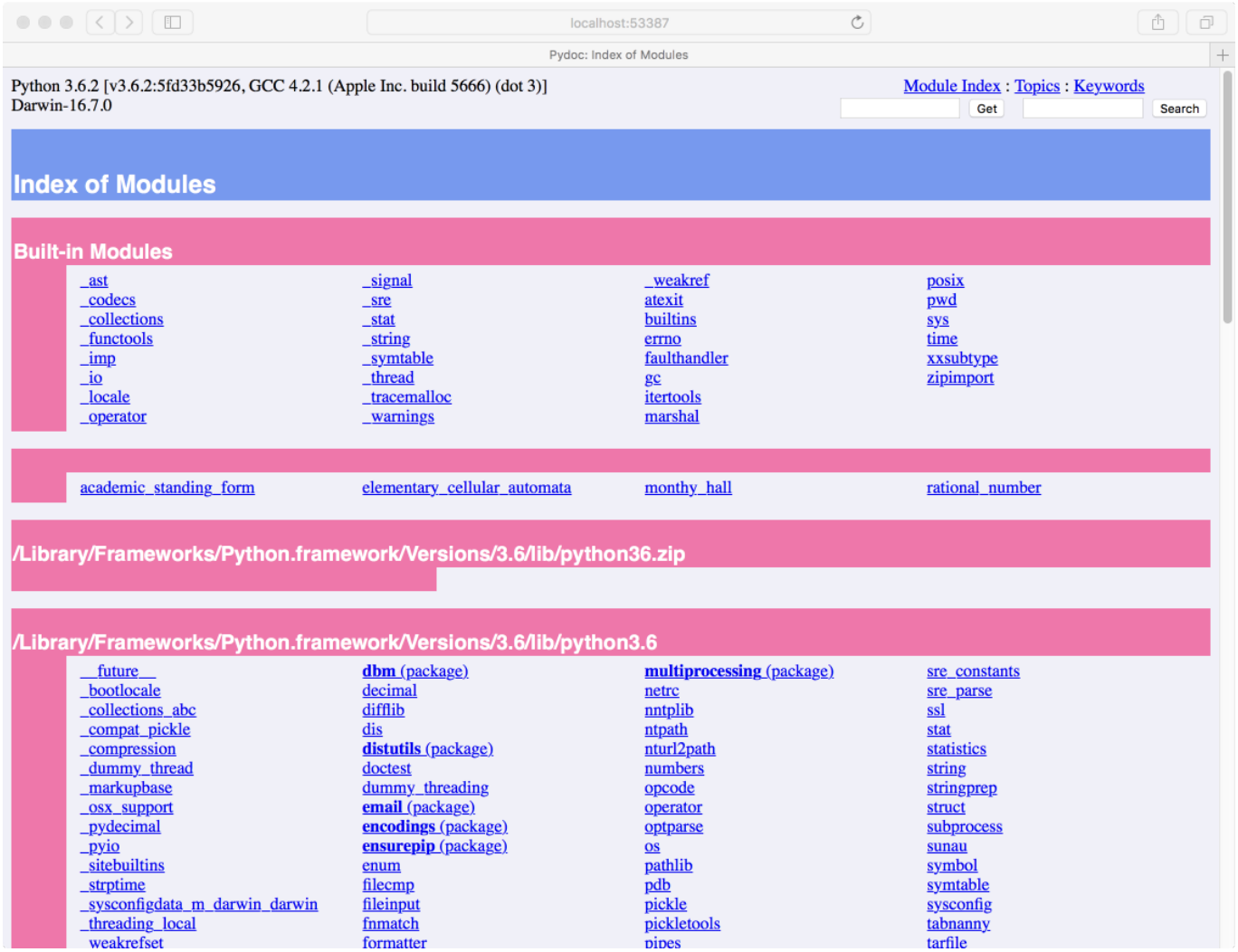
13. Help on a name in a module thanks to doctrings

```
>>> import rational_number
>>> help(rational_number.compute_fraction)
Help on function compute_fraction in module rational_number:

compute_fraction(sigma, tau)
    Based on the computation
            0.(sigma)(tau)(tau)(tau)...
          = sigma * 10^{-|sigma|} + tau(10^{-|sigma|-|tau|} +
                                        10^{-|sigma|-2|tau|} +
                                        ...)
          = sigma * 10^{-|sigma|} +
            tau * 10^{-|sigma|-|tau|} / (1 - 10^{-|tau|})
          = sigma * 10^{-|sigma|} +
            tau * 10^{-|sigma|} / (10^{|tau|} - 1)
          = [sigma * 10^{-|sigma|} * (10^{|tau|} - 1) + tau * 10^{-|sigma|}] /
            (10^{|tau|} - 1)
          = [sigma * (10^{|tau|} - 1) + tau] /
            [(10^{|tau|} - 1) * 10^{|sigma|}]

    >>> compute_fraction('0', '0')
    (0, 90)
    >>> compute_fraction('0', '1')
    (1, 90)
    >>> compute_fraction('1', '0')
    (9, 90)
...
>>> from rational_number import compute_fraction
>>> help(compute_fraction)
Help on function compute_fraction in module rational_number:

compute_fraction(sigma, tau)
    Based on the computation
            0.(sigma)(tau)(tau)(tau)...
          = sigma * 10^{-|sigma|} + tau(10^{-|sigma|-|tau|} +
                                        10^{-|sigma|-2|tau|} +
                                        ...)
          = sigma * 10^{-|sigma|} +
            tau * 10^{-|sigma|-|tau|} / (1 - 10^{-|tau|})
          = sigma * 10^{-|sigma|} +
            tau * 10^{-|sigma|} / (10^{|tau|} - 1)
          = [sigma * 10^{-|sigma|} * (10^{|tau|} - 1) + tau * 10^{-|sigma|}] /
            (10^{|tau|} - 1)
          = [sigma * (10^{|tau|} - 1) + tau] /
            [(10^{|tau|} - 1) * 10^{|sigma|}]

    >>> compute_fraction('0', '0')
    (0, 90)
    >>> compute_fraction('0', '1')
    (1, 90)
    >>> compute_fraction('1', '0')
    (9, 90)
...
```

Part 3. Help in a browser with PyDoc

Includes documentation for the scripts in the directory where the command is executed.

```
$ python3 -m pydoc -b
```



Part 4. Unit testing of functions from modules we create

Made possible thanks to:

```
if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

14. Passing all tests

```
$ python3 rational_number.py
$ python3 rational_number.py -v
Trying:
    compute_fraction('0', '0')
Expecting:
    (0, 90)
ok
Trying:
    compute_fraction('0', '1')
Expecting:
    (1, 90)
ok
Trying:
    compute_fraction('1', '0')
Expecting:
    (9, 90)
ok
Trying:
    compute_fraction('1', '1')
Expecting:
    (10, 90)
ok
Trying:
    compute_fraction('9', '9')
Expecting:
    (90, 90)
ok
...
5 items had no tests:
    __main__
    __main__.determine_reduced_fraction_from_pattern_and_repeated_pattern
    __main__.get
    __main__.get_both_inputs
    __main__.output_result
4 items passed all tests:
  11 tests in __main__.compute_fraction
  11 tests in __main__.determine_reduced_fraction
  11 tests in __main__.gcd
  11 tests in __main__.reduce_fraction
44 tests in 9 items.
44 passed and 0 failed.
Test passed.
```

15. Failing some tests

Changing

```
    >>> reduce_fraction(0, 1)
    (0, 1)
```

to

```
    >>> reduce_fraction(0, 1)
    (1, 1)
```

in file.

```
$ python3 rational_number.py
**********************************************************************
File "rational_number.py", line 89, in __main__.reduce_fraction
Failed example:
    reduce_fraction(0, 1)
Expected:
    (1, 1)
Got:
    (0, 1)
**********************************************************************
1 items had failures:
   1 of  11 in __main__.reduce_fraction
***Test Failed*** 1 failures.
```