DS-210: Programming for Data Science

Lecture 13: Documentation generation in Python. Version control.

What are the goals of documentation?

What are the goals of documentation?

- **Users:** how to use your software
- **Developers:** how to extend it and maintain it

Challenges of good documentation

Keeping it

- up to date
- concise
- exhaustive

Challenges of good documentation

Keeping it

- up to date
- concise
- exhaustive

Various types

- Code commenting
- Reference
- Tutorial
- Quick HOWTO's
- ...

A few tips on code commenting

- Make code explain itself as much as possible
    - Code refactoring is very important
- Explain only non-trivial parts
    - The reader should already know this programming language
- Keep comments close to the related code

In [ ]:
```
# DON'T DO THIS!!!
def add(x,y):
    # we add the first parameter to the second one and return their sum
    return x + y
```

A few tips on code commenting

- Make code explain itself as much as possible
    - Code refactoring is very important
- Explain only non-trivial parts
    - The reader should already know this programming language
- Keep comments close to the related code

In [ ]:
```
# DON'T DO THIS!!!
def add(x,y):
    # we add the first parameter to the second one and return their sum
    return x + y
```

- Can be used for tagging parts of the code (`TODO` or `FIXME`)

In [ ]:
```
def return_one(x):
    # FIXME: properly handle the corner case of x = 0
    return x/x
```

Today's focus: docstrings

Main idea:

- put the description of a function, class, or object in a comment block next to it

Advantages:

- for someone reading or maintaining the code: you'll find your information right there
- encourages the developer to update it after making changes
- can be extracted automatically to produce nice docs (we'll look at examples today)

```
In [ ]: def addition(x,y):
            """(one-line summary) Addition of two numbers.

            Here is where you can get into more details. `x`
            is an integer and so is `y`. The output is the
            sum of `x` and `y`.
            """
            return x + y
```

Today's focus: docstrings

Main idea:

- put the description of a function, class, or object in a comment block next to it

```
In [ ]: def addition(x,y):
            """(one-line summary) Addition of two numbers.

            Here is where you can get into more details. `x`
            is an integer and so is `y`. The output is the
            sum of `x` and `y`.
            """
            return x + y
```

Advantages:

- for someone reading or maintaining the code: you'll find your information right there
- encourages the developer to update it after making changes
- can be extracted automatically to produce nice docs (we'll look at examples today)

Popular in many languages:

- easy to add even if officially not supported

Simplest tool: `pydoc`

- Should be included in your Python installation
- Go to the command line and just type: `pydoc WHATEVER-YOU-ARE-INTERESTED-IN`
- Example: `pydoc matplotlib.pyplot.scatter` or `pydoc numpy.array`

File Edit View Search Terminal Help

```
[user@jupyter-notebook ~]$ pydoc numpy.array
```

```
Help on built-in function array in numpy:

numpy.array = array(...)
    array(object, dtype=None, *, copy=True, order='K', subok=False, ndmin=0,
          like=None)

    Create an array.

    Parameters
    ----------
    object : array_like
        An array, any object exposing the array interface, an object whose
        __array__ method returns an array, or any (nested) sequence.
    dtype : data-type, optional
        The desired data-type for the array.  If not given, then the type will
        be determined as the minimum type required to hold the objects in the
        sequence.
    copy : bool, optional
        If true (default), then the object is copied.  Otherwise, a copy will
        only be made if __array__ returns a copy, if obj is a nested sequence,
        or if a copy is needed to satisfy any of the other requirements
        (`dtype`, `order`, etc.).
    order : {'K', 'A', 'C', 'F'}, optional
        Specify the memory layout of the array. If object is not an array, the
        newly created array will be in C order (row major) unless 'F' is
        specified, in which case it will be in Fortran order (column major).
        If object is an array the following holds.

        ===== ========= ===================================================
        order  no copy                     copy=True
        ===== ========= ===================================================
        'K'   unchanged F & C order preserved, otherwise most similar order
        'A'   unchanged F order if input is F and not C, otherwise C order
        'C'   C order   C order
        'F'   F order   F order
        ===== ========= ===================================================

        When ``copy=False`` and a copy is made for other reasons, the result is
        the same as if ``copy=True``, with some exceptions for `A`, see the
        Notes section. The default order is 'K'.
    subok : bool, optional
:
```

Using `pdoc` for generating HTML/Markdown/PDF

Installation: `pip install pdoc3` (adjust to your package manager!)

Generating HTML pages: `pdoc --html MODULE-NAME` (with `--force` to overwrite an earlier version)

Using `pdoc` for generating HTML/Markdown/PDF

Installation: `pip install pdoc3` (adjust to your package manager!)

Generating HTML pages: `pdoc --html MODULE-NAME` (with `--force` to overwrite an earlier version)

[Demo in the terminal]

File Edit View Search Terminal Help

```
[user@jupyter-notebook example]$ tree
.
├── examples.py
└── varia
    ├── __init__.py
    ├── add.py
    ├── emotions.py
    └── multiply.py

1 directory, 5 files
[user@jupyter-notebook example]$
```

File Edit View Search Terminal Help

```
[user@jupyter-notebook example]$ tree
.
├── examples.py
└── varia
    ├── __init__.py
    ├── add.py
    ├── emotions.py
    └── multiply.py

1 directory, 5 files
[user@jupyter-notebook example]$ cat examples.py
import varia.multiply as mult
from varia.add import double

import varia.emotions as emo

if __name__ == "__main__":
    x = 7
    y = 9
    print(x,"times",y,"is",mult.multiply(x,y))
    print('doubled',x,'is',double(x))

    print('====================')

    s = 'angry Bob'
    t = 'surprised Alice'

    print(s,'and',t)

    s = emo.make_sad(s)
    t = emo.make_happy(t)

    print(s,'and',t)
[user@jupyter-notebook example]$ python examples.py
7 times 9 is 63
doubled 7 is 14
====================
angry Bob and surprised Alice
sad Bob and happy Alice
[user@jupyter-notebook example]$
```

File Edit View Search Terminal Help

```
[user@jupyter-notebook example]$ cd varia/
[user@jupyter-notebook varia]$ view __init__.py
```

File Edit View Search Terminal Help

```
"""One-line summary: lots of things.

Now a longer description follows. Here you can write a lot. Honestly, you
shouldn't be putting all these packages together because they have nothing in
common. But at least you can use markdown here: *italics* **bold**. My favorite
variable name is `nothing`. By the way, some people claim that this is the most
beautiful equation: \[e^{i\pi} + 1 = 0\]

.. warning::
    The interface here is unstable.
"""
```

"__init__.py" [readonly] 11L, 445B 7,38 All

File Edit View Search Terminal Help

```
[user@jupyter-notebook example]$ cd varia/
[user@jupyter-notebook varia]$ view __init__.py
[user@jupyter-notebook varia]$ view add.py
```

File Edit View Search Terminal Help

```
"""This module handles things related to addition

This is a longer description.
"""

def add(x, y, z=0):
    """Addition of up to three numbers

    Arguments:

      `x`: first term to add

      `y`: second term to add

      `z`: third term to add (optional, 0 if not present)

    Returns:
      The sum of `x`, `y`, and `z`, i.e., \(x+y+z\) or \(\ln(e^x \cdot e^y \cdot e^z)\).
    """

    return x + y + z

def double(x):
    """Doubles its input.

    .. depracated::
       Just multiply by two instead.

    """
    return x + x
~
```

"add.py" [readonly] 30L, 572B 15,22 All

File Edit View Search Terminal Help

```
[user@jupyter-notebook example]$ cd varia/
[user@jupyter-notebook varia]$ view __init__.py
[user@jupyter-notebook varia]$ view add.py
[user@jupyter-notebook varia]$ view multiply.py
```

File Edit View Search Terminal Help

```
"""Compute various functions related to multiplication.
"""

def multiply(x,y):
    """Multiply two numbers.

    Parameters
    ----------
    x: number
        the first number to be multiplied
    y: number
        the second number to be multiplied

    Returns
    -------
    number
        the product of the numbers


    .. danger::
        Multiplying by zero is irreversible.
    """
    return x * y

def square(x):
    """Compute a square of a number.

    Parameters
    ----------
    x: number
        the number to be squared

    Returns
    -------
    number
        the square of `x` (i.e., \(x^2\))
    """
    return x * x

def cube(x):
    """Compute a cube of a number.

    Parameters
    ----------
    x: number
        the number to be squared

    Returns
    -------
    number
        the cube of `x` (i.e., \(x^3\))
    """
    return square(x) * x
~
~
~
~
~
~
~
```

"multiply.py" [readonly] 53L, 882B 31,7 All

File Edit View Search Terminal Help

```
[user@jupyter-notebook example]$ cd varia/
[user@jupyter-notebook varia]$ view __init__.py
[user@jupyter-notebook varia]$ view add.py
[user@jupyter-notebook varia]$ view multiply.py
[user@jupyter-notebook varia]$ view emotions.py
```

```
"""Package with functions for operating on emotions in strings.

Not very useful and doesn't handle that many real-world cases.
"""

# note that __EMOTIONS__ and _set_emotions do not make it to the library
__EMOTIONS__ = ['happy','sad','angry','surprised']

def remove_emotions(description):
    """Remove prefixes corresponding to emotions.

    Args:
        descripion (str): The string to be processed

    Returns:
        str: The input string without prefixes corresponding to emotions if there are any.
    """

    global __EMOTIONS__
    for prefix in __EMOTIONS__:
        if description.startswith(prefix + ' '):
            # recurse in case there is another emotion
            return remove_emotions(description[len(prefix)+1:])
    return description

def _set_emotion(description, emotion, capitalize):
    tmp = emotion + ' ' + remove_emotions(description)
    return tmp.capitalize() if capitalize else tmp

def make_angry(description, capitalize = False):
    """Make the described object **angry.**

    This is an example of a Google Docstring formatting.

    Args:
        description (str): The string to be processed
        capitalize (bool): Whether the first character in the output should
            be capitalized (default is False)

    Returns:
        str: The input string without initial emotions and with the prefix `angry` instead.
    """
    return _set_emotion(description, 'angry', capitalize)

def make_surprised(description, capitalize = False):
    """Make the described object **surprised.**

    This is an example of a numpy doc formatting.

    Parameters
    ----------
    description: str
        The string to be processed
    capitalize: bool, optional
        Whether the first character in the output should be capitalized (default is False)

    Returns
    -------
    str
        The input string without initial emotions and with the prefix `surprised` instead.
```

```
        capitalize (bool): Whether the first character in the output should
            be capitalized (default is False)

    Returns:
        str: The input string without initial emotions and with the prefix `angry` instead.
    """
    return _set_emotion(description, 'angry', capitalize)

def make_surprised(description, capitalize = False):
    """Make the described object **surprised.**

    This is an example of a numpy doc formatting.

    Parameters
    ----------
    description: str
        The string to be processed
    capitalize: bool, optional
        Whether the first character in the output should be capitalized (default is False)

    Returns
    -------
    str
        The input string without initial emotions and with the prefix `surprised` instead.
    """
    return _set_emotion(description, 'surprised', capitalize)

def make_happy(description, capitalize = False):
    """Make the described object **happy.**

    This is an example of a Google Docstring formatting.

    Args:
        description (str): The string to be processed
        capitalize (bool): Whether the first character in the output should
            be capitalized (default is False)

    Returns:
        str: The input string without initial emotions and with the prefix `happy` instead.
    """
    return _set_emotion(description, 'happy', capitalize)

def make_sad(description, capitalize = False):
    """Make the described object **sad.**

    This is an example of a numpy doc formatting.

    Parameters
    ----------
    description: str
        The string to be processed
    capitalize: bool, optional
        Whether the first character in the output should be capitalized (default is False)

    Returns
    -------
    str
        The input string without initial emotions and with the prefix `sad` instead.
    """
    return _set_emotion(description, 'sad', capitalize)
```

File Edit View Search Terminal Help

```
[user@jupyter-notebook varia]$ cd ..
```

File Edit View Search Terminal Help

```
[user@jupyter-notebook varia]$ cd ..
[user@jupyter-notebook example]$ tree
.
├── examples.py
└── varia
    ├── __init__.py
    ├── __pycache__
    │   ├── __init__.cpython-39.pyc
    │   ├── add.cpython-39.pyc
    │   ├── emotions.cpython-39.pyc
    │   └── multiply.cpython-39.pyc
    ├── add.py
    ├── emotions.py
    └── multiply.py

2 directories, 9 files
[user@jupyter-notebook example]$
```

File Edit View Search Terminal Help

```
[user@jupyter-notebook varia]$ cd ..
[user@jupyter-notebook example]$ tree
.
├── examples.py
└── varia
    ├── __init__.py
    ├── __pycache__
    │   ├── __init__.cpython-39.pyc
    │   ├── add.cpython-39.pyc
    │   ├── emotions.cpython-39.pyc
    │   └── multiply.cpython-39.pyc
    ├── add.py
    ├── emotions.py
    └── multiply.py

2 directories, 9 files
[user@jupyter-notebook example]$ pdoc --html --config latex_math=True varia
<unknown>:1: DeprecationWarning: invalid escape sequence \[
<unknown>:26: DeprecationWarning: invalid escape sequence \(
html/varia/index.html
html/varia/add.html
html/varia/emotions.html
html/varia/multiply.html
[user@jupyter-notebook example]$
```

File Edit View Search Terminal Help

```
[user@jupyter-notebook varia]$ cd ..
[user@jupyter-notebook example]$ tree
.
├── examples.py
└── varia
    ├── __init__.py
    ├── __pycache__
    │   ├── __init__.cpython-39.pyc
    │   ├── add.cpython-39.pyc
    │   ├── emotions.cpython-39.pyc
    │   └── multiply.cpython-39.pyc
    ├── add.py
    ├── emotions.py
    └── multiply.py

2 directories, 9 files
[user@jupyter-notebook example]$ pdoc --html --config latex_math=True varia
<unknown>:1: DeprecationWarning: invalid escape sequence \[
<unknown>:26: DeprecationWarning: invalid escape sequence \(
html/varia/index.html
html/varia/add.html
html/varia/emotions.html
html/varia/multiply.html
[user@jupyter-notebook example]$ tree
.
├── examples.py
├── html
│   └── varia
│       ├── add.html
│       ├── emotions.html
│       ├── index.html
│       └── multiply.html
└── varia
    ├── __init__.py
    ├── __pycache__
    │   ├── __init__.cpython-39.pyc
    │   ├── add.cpython-39.pyc
    │   ├── emotions.cpython-39.pyc
    │   └── multiply.cpython-39.pyc
    ├── add.py
    ├── emotions.py
    └── multiply.py

4 directories, 13 files
[user@jupyter-notebook example]$
```

Package **varia**

One-line summary: lots of things.

Now a longer description follows. Here you can write a lot. Honestly, you shouldn't be putting all these packages together because they have nothing in common. But at least you can use markdown here: *italics* **bold**. My favorite variable name is `nothing`. By the way, some people claim that this is the most beautiful equation:

$$e^{i\pi} + 1 = 0$$

Warning

The interface here is unstable.

▸ EXPAND SOURCE CODE

Sub-modules

varia.add

This module handles things related to addition …

varia.emotions

Package with functions for operating on emotions in strings …

varia.multiply

Compute various functions related to multiplication.

Index

Super-module

Functions

Module **varia.add**

This module handles things related to addition

This is a longer description.

▸ EXPAND SOURCE CODE

Functions

```
def add(x, y, z=0)
```

Addition of up to three numbers

Arguments

`x` : first term to add

`y` : second term to add

`z` : third term to add (optional, 0 if not present)

Returns

The sum of `x`, `y`, and `z`, i.e., $x + y + z$ or $\ln(e^x \cdot e^y \cdot e^z)$.

▸ EXPAND SOURCE CODE

```
def double(x)
```

Doubles its input.

Depracated

Just multiply by two instead.

▸ EXPAND SOURCE CODE

Index

Super-module

Functions

Module **varia.multiply**

Compute various functions related to multiplication.

▸ EXPAND SOURCE CODE

Functions

```
def cube(x)
```

Compute a cube of a number.

Parameters

x : number
the number to be squared

Returns

number
the cube of `x` (i.e., x^3)

▸ EXPAND SOURCE CODE

```
def multiply(x, y)
```

Multiply two numbers.

Parameters

x : number
the first number to be multiplied

y : number
the second number to be multiplied

Returns

number
the product of the numbers

Danger

Multiplying by zero is irreversible.

Module **varia.emotions**

Package with functions for operating on emotions in strings.

Not very useful and doesn't handle that many real-world cases.

▸ EXPAND SOURCE CODE

Functions

```
def make_angry(description, capitalize=False)
```

Make the described object **angry.**

This is an example of a Google Docstring formatting.

Args

description : `str`
The string to be processed

capitalize : `bool`
Whether the first character in the output should be capitalized (default is False)

Returns

`str`
The input string without initial emotions and with the prefix `angry` instead.

▸ EXPAND SOURCE CODE

```
def make_happy(description, capitalize=False)
```

Make the described object **happy.**

This is an example of a Google Docstring formatting.

Args

description : `str`
The string to be processed

capitalize : `bool`

Many other tools for processing docstrings in Python

- Explore them before committing to one of them
- Check out Sphinx!

Version control

- Access to past versions
- Make branches and develop new features independently
- Merge them when you are ready

Version control

- Access to past versions
- Make branches and develop new features independently
- Merge them when you are ready

Git

- **[Quick demo in the terminal]**
- GUI tools available as well
- Most popular these days
- Can be used for distributed collaboration
- Popular hosting sites: Github, Gitlab, ..

Discussion tomorrow: quick demo of Github

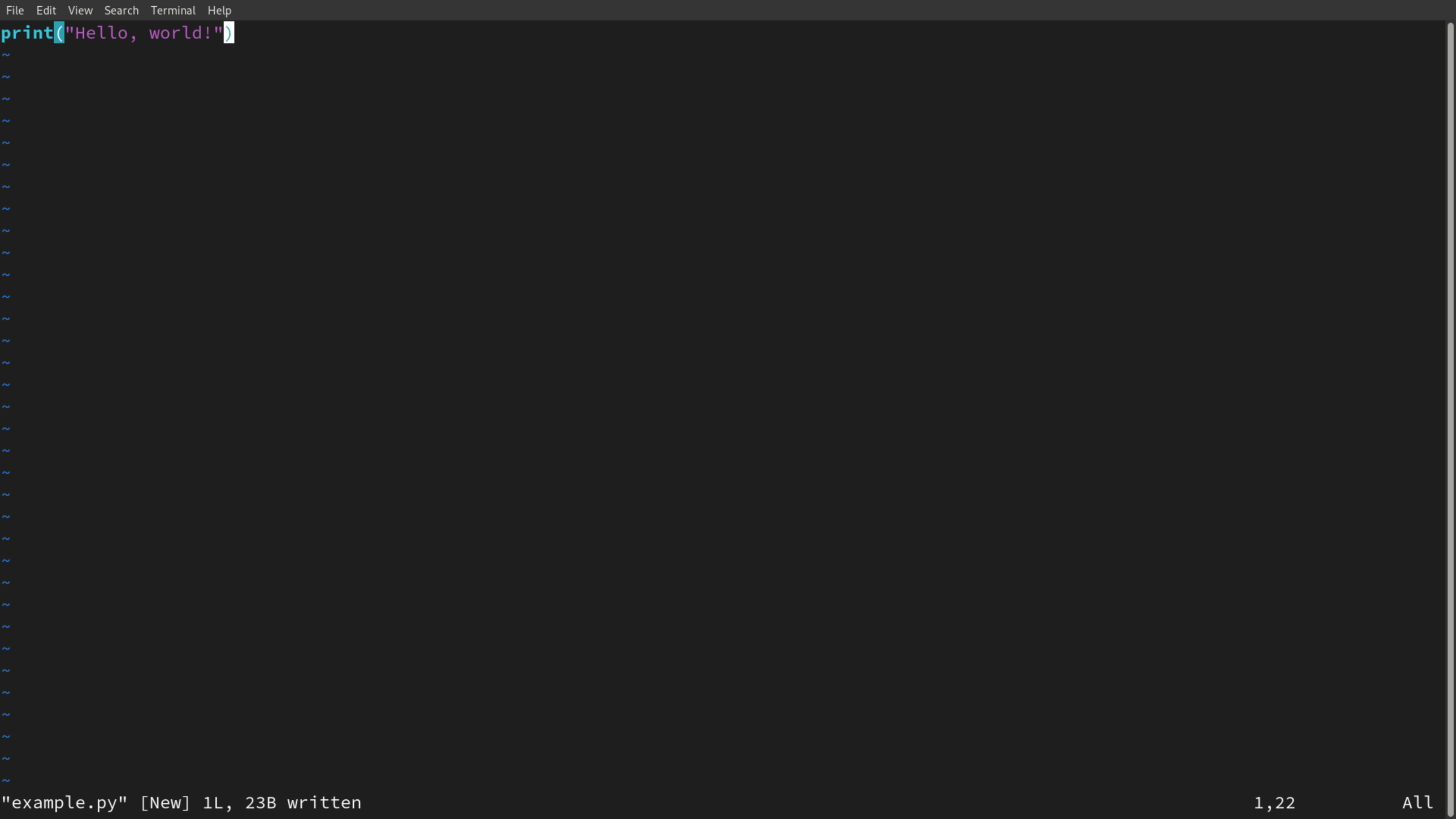
File Edit View Search Terminal Help

```
[user@jupyter-notebook ~]$ git init git-demo
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/user/git-demo/.git/
[user@jupyter-notebook ~]$
```

File Edit View Search Terminal Help

```
[user@jupyter-notebook ~]$ git init git-demo
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/user/git-demo/.git/
[user@jupyter-notebook ~]$ cd git-demo/
[user@jupyter-notebook git-demo]$
```

```
[user@jupyter-notebook ~]$ git init git-demo
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/user/git-demo/.git/
[user@jupyter-notebook ~]$ cd git-demo/
[user@jupyter-notebook git-demo]$ git config --global user.name "Martin Luther King Jr."
[user@jupyter-notebook git-demo]$ git config --global user.email "mlk@bu.edu"
[user@jupyter-notebook git-demo]$
```


File Edit View Search Terminal Help
print("Hello, world!")
"example.py" [New] 1L, 23B written
1,22
All

```
[user@jupyter-notebook ~]$ git init git-demo
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/user/git-demo/.git/
[user@jupyter-notebook ~]$ cd git-demo/
[user@jupyter-notebook git-demo]$ git config --global user.name "Martin Luther King Jr."
[user@jupyter-notebook git-demo]$ git config --global user.email "mlk@bu.edu"
[user@jupyter-notebook git-demo]$ vim example.py
[user@jupyter-notebook git-demo]$ git add -v example.py
add 'example.py'
[user@jupyter-notebook git-demo]$
```

```
File  Edit  View  Search  Terminal  Help
[user@jupyter-notebook ~]$ git init git-demo
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/user/git-demo/.git/
[user@jupyter-notebook ~]$ cd git-demo/
[user@jupyter-notebook git-demo]$ git config --global user.name "Martin Luther King Jr."
[user@jupyter-notebook git-demo]$ git config --global user.email "mlk@bu.edu"
[user@jupyter-notebook git-demo]$ vim example.py
[user@jupyter-notebook git-demo]$ git add -v example.py
add 'example.py'
[user@jupyter-notebook git-demo]$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   example.py

[user@jupyter-notebook git-demo]$ git commit -vm "initial version with one hello"
[master (root-commit) 86aec64] initial version with one hello
 1 file changed, 1 insertion(+)
 create mode 100644 example.py
[user@jupyter-notebook git-demo]$
```

File Edit View Search Terminal Help

```
[user@jupyter-notebook git-demo]$ vim example.py
```

File Edit View Search Terminal Help

```
print("Hello, world!")
print("And hello, BU!")
```

"example.py" 2L, 47B written 2,23 All

File Edit View Search Terminal Help

```
[user@jupyter-notebook git-demo]$ vim example.py
[user@jupyter-notebook git-demo]$ python example.py
Hello, world!
And hello, BU!
[user@jupyter-notebook git-demo]$
```

```
File  Edit  View  Search  Terminal  Help
[user@jupyter-notebook git-demo]$ vim example.py
[user@jupyter-notebook git-demo]$ python example.py
Hello, world!
And hello, BU!
[user@jupyter-notebook git-demo]$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   example.py

no changes added to commit (use "git add" and/or "git commit -a")
[user@jupyter-notebook git-demo]$
```

```
File  Edit  View  Search  Terminal  Help
[user@jupyter-notebook git-demo]$ vim example.py
[user@jupyter-notebook git-demo]$ python example.py
Hello, world!
And hello, BU!
[user@jupyter-notebook git-demo]$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   example.py

no changes added to commit (use "git add" and/or "git commit -a")
[user@jupyter-notebook git-demo]$ git commit -vam "adding another hello"
[master 2a7d58f] adding another hello
 1 file changed, 1 insertion(+)
[user@jupyter-notebook git-demo]$
```

```
File  Edit  View  Search  Terminal  Help
[user@jupyter-notebook git-demo]$ vim example.py
[user@jupyter-notebook git-demo]$ python example.py
Hello, world!
And hello, BU!
[user@jupyter-notebook git-demo]$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   example.py

no changes added to commit (use "git add" and/or "git commit -a")
[user@jupyter-notebook git-demo]$ git commit -vam "adding another hello"
[master 2a7d58f] adding another hello
 1 file changed, 1 insertion(+)
[user@jupyter-notebook git-demo]$ git log
commit 2a7d58f00c66edd8d85aff2762299ea0463c006b (HEAD -> master)
Author: Martin Luther King Jr <mlk@bu.edu>
Date:   Tue Feb 22 17:03:12 2022 -0500

    adding another hello

commit 86aec64c684f27019f073dc2081298fb11f11109
Author: Martin Luther King Jr <mlk@bu.edu>
Date:   Tue Feb 22 16:57:06 2022 -0500

    initial version with one hello
[user@jupyter-notebook git-demo]$
```

File Edit View Search Terminal Help

```
[user@jupyter-notebook git-demo]$ git branch third-hello
[user@jupyter-notebook git-demo]$ git switch third-hello
Switched to branch 'third-hello'
[user@jupyter-notebook git-demo]$ vim example.py
```

File Edit View Search Terminal Help

```
print("Hello, world!")
print("And hello, BU!")
print("And hello, Boston!")
```

"example.py" 3L, 75B written 3,27 All

File Edit View Search Terminal Help

```
[user@jupyter-notebook git-demo]$ git branch third-hello
[user@jupyter-notebook git-demo]$ git switch third-hello
Switched to branch 'third-hello'
[user@jupyter-notebook git-demo]$ vim example.py
[user@jupyter-notebook git-demo]$ git status
On branch third-hello
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   example.py

no changes added to commit (use "git add" and/or "git commit -a")
[user@jupyter-notebook git-demo]$ git commit -vam "adding a third hello"
[third-hello ade2a77] adding a third hello
 1 file changed, 1 insertion(+)
[user@jupyter-notebook git-demo]$
```

File Edit View Search Terminal Help

```
[user@jupyter-notebook git-demo]$ git switch master
Switched to branch 'master'
[user@jupyter-notebook git-demo]$ cat example.py
print("Hello, world!")
print("And hello, BU!")
[user@jupyter-notebook git-demo]$
```

```
[user@jupyter-notebook git-demo]$ git switch master
Switched to branch 'master'
[user@jupyter-notebook git-demo]$ cat example.py
print("Hello, world!")
print("And hello, BU!")
[user@jupyter-notebook git-demo]$ git merge third-hello
Updating 2a7d58f..ade2a77
Fast-forward
 example.py | 1 +
 1 file changed, 1 insertion(+)
[user@jupyter-notebook git-demo]$ cat example.py
print("Hello, world!")
print("And hello, BU!")
print("And hello, Boston!")
[user@jupyter-notebook git-demo]$
```

File Edit View Search Terminal Help

```
[user@jupyter-notebook git-demo]$ git log
commit ade2a7728711cdf32689a5cf11f8b4a05755b479 (HEAD -> master, third-hello)
Author: Martin Luther King Jr <mlk@bu.edu>
Date:   Tue Feb 22 17:04:58 2022 -0500

    adding a third hello

commit 2a7d58f00c66edd8d85aff2762299ea0463c006b
Author: Martin Luther King Jr <mlk@bu.edu>
Date:   Tue Feb 22 17:03:12 2022 -0500

    adding another hello

commit 86aec64c684f27019f073dc2081298fb11f11109
Author: Martin Luther King Jr <mlk@bu.edu>
Date:   Tue Feb 22 16:57:06 2022 -0500

    initial version with one hello
[user@jupyter-notebook git-demo]$
```

```
Author: Martin Luther King Jr <mlk@bu.edu>
Date:   Tue Feb 22 17:04:58 2022 -0500

    adding a third hello

commit 2a7d58f00c66edd8d85aff2762299ea0463c006b
Author: Martin Luther King Jr <mlk@bu.edu>
Date:   Tue Feb 22 17:03:12 2022 -0500

    adding another hello

commit 86aec64c684f27019f073dc2081298fb11f11109
Author: Martin Luther King Jr <mlk@bu.edu>
Date:   Tue Feb 22 16:57:06 2022 -0500

    initial version with one hello
[user@jupyter-notebook git-demo]$ git checkout 86aec64c684f27019f073dc2081298fb11f11109
Note: switching to '86aec64c684f27019f073dc2081298fb11f11109'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 86aec64 initial version with one hello
[user@jupyter-notebook git-demo]$
```

File Edit View Search Terminal Help

```
[user@jupyter-notebook git-demo]$ cat example.py
print("Hello, world!")
[user@jupyter-notebook git-demo]$ git checkout master
Previous HEAD position was 86aec64 initial version with one hello
Switched to branch 'master'
[user@jupyter-notebook git-demo]$ cat example.py
print("Hello, world!")
print("And hello, BU!")
print("And hello, Boston!")
[user@jupyter-notebook git-demo]$
```

File Edit View Search Terminal Help

```
[user@jupyter-notebook git-demo]$ cat example.py
print("Hello, world!")
[user@jupyter-notebook git-demo]$ git checkout master
Previous HEAD position was 86aec64 initial version with one hello
Switched to branch 'master'
[user@jupyter-notebook git-demo]$ cat example.py
print("Hello, world!")
print("And hello, BU!")
print("And hello, Boston!")
[user@jupyter-notebook git-demo]$
```