Installing miniconda (the quick version for macOS)

- get and install

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh
chmod 0755 Miniconda3-latest-MacOSX-x86_64.sh
./Miniconda3-latest-MacOSX-x86_64.sh
# answer questions it asks you

# the next line reloads your bash profile your CLI environment
# knows where this new python lives
source ~/.bash_profile
```

- test to make sure you have the correct python

```
which python
# should look something like the following
/Users/<you>/miniconda3/bin/python

python --version
# should look something like the following
Python 3.6.4 :: Anaconda custom (64-bit)
```

Running python from the command-line

- by default, if you install conda3, you get python 3.6 (or 3.7):

```
which python
python --version
```

- we can create environments within conda for different pythons (if we need python 2.7):

```
conda create -n class_py27 python=2.7
```

- each environment that you create (like above) is isolated from on another, so if we create:

```
conda create -n class_test1
conda install requests
# deactivate this environment
source deactivate

conda create -n class_test2
conda install numpy
# deactivate this environment
source deactivate
```

These two package installations will be isolated from one another (e.g. numpy is only available in the class_test2 environment).

- ok, now... how to run programs. let's make sure we're in default conda environment

```
source deactivate
```

- now, go to some location on your filesystem

```
mkdir -p $HOME/tmp
cd $HOME/tmp
```

- create a new python program in a file editor (or using vim, etc) and paste the following in it:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

def main():
    print("Hello world!")

if __name__ == '__main__':
    main()
```

- save that file in your newly created directory as `program1.py`
- run it with:

```
python program1.py
```

Getting arguments into python - approach 1

Sometimes to need to get arguments into python. There are ugly and pretty ways to do this. An ugly way uses `sys.argv`.

- Create a new file called `program2.py`

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys

def main():
    print("These are my arguments:")
    print("The first argument in sys.argv is the program name", sys.argv[0])
    print("The rest are parsed as a list:", sys.argv[1:])

if __name__ == '__main__':
    main()
```

- Run this program:

```
python program2.py
```

- Now, run it with some arguments

```
python program2.py value number character
```

- You can access these arguments now in your program:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys

def main():
    print("These are my arguments:")
    print("The first argument in sys.argv is the program name", sys.argv[0])
    print("The rest are parsed as a list:", sys.argv[1:])
    print("Here's the first argument:\t", sys.argv[1])
    print("Here's the second argument:\t", sys.argv[2])
    print("Here's the third argument:\t", sys.argv[3])

if __name__ == '__main__':
    main()
```

Getting arguments into python - approach 2

So, the above option works, but it's a little coarse - how to you specify the types of arguments you want from a user (e.g. integer values, etc). That's where the python module argparse comes into play...

- Create the following as `program3.py`:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import argparse

def get_args():
    parser = argparse.ArgumentParser(description='My program does some stuff.')
    # this is a non-optional argument with no flag
    parser.add_argument(
        'integer',
        type=int,
        help='some integer value'
    )
    parser.add_argument(
        '--flagged-arg',
        type=str,
        help='heres an argument with a flag'
    )
    return parser.parse_args()


def main():
    args = get_args()
    print("These are all the args: ", args)
    print("And this is how we access the arg for integer: ", args.integer)

if __name__ == '__main__':
    main()
```

- Sometimes we want to make flagged options non-optional (i find them clearer to use). Create `program4.py`:
- Create the following as `program3.py`:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import argparse

def get_args():
    parser = argparse.ArgumentParser(description='My program does some stuff.')
    # this is a non-optional argument with no flag
    parser.add_argument(
        '--integer',
        type=int,
        required=True,
        help='some integer value'
    )
    parser.add_argument(
        '--flagged-arg',
```

```
        type=str,
        required=True,
        help='heres an argument with a flag'
    )
    return parser.parse_args()


def main():
    args = get_args()
    print("These are all the args: ", args)
    print("And this is how we access the arg for integer: ", args.integer)
    print("And this is how we access the arg for flagged-arg: ", args.flagged_arg)

if __name__ == '__main__':
    main()
```