Basic Python

~

Why python?

- All the cool kids are using it- lots of great scientific analysis packages
- Businesses use it too!
- Open source and lots of online support forums

Programming advice #1

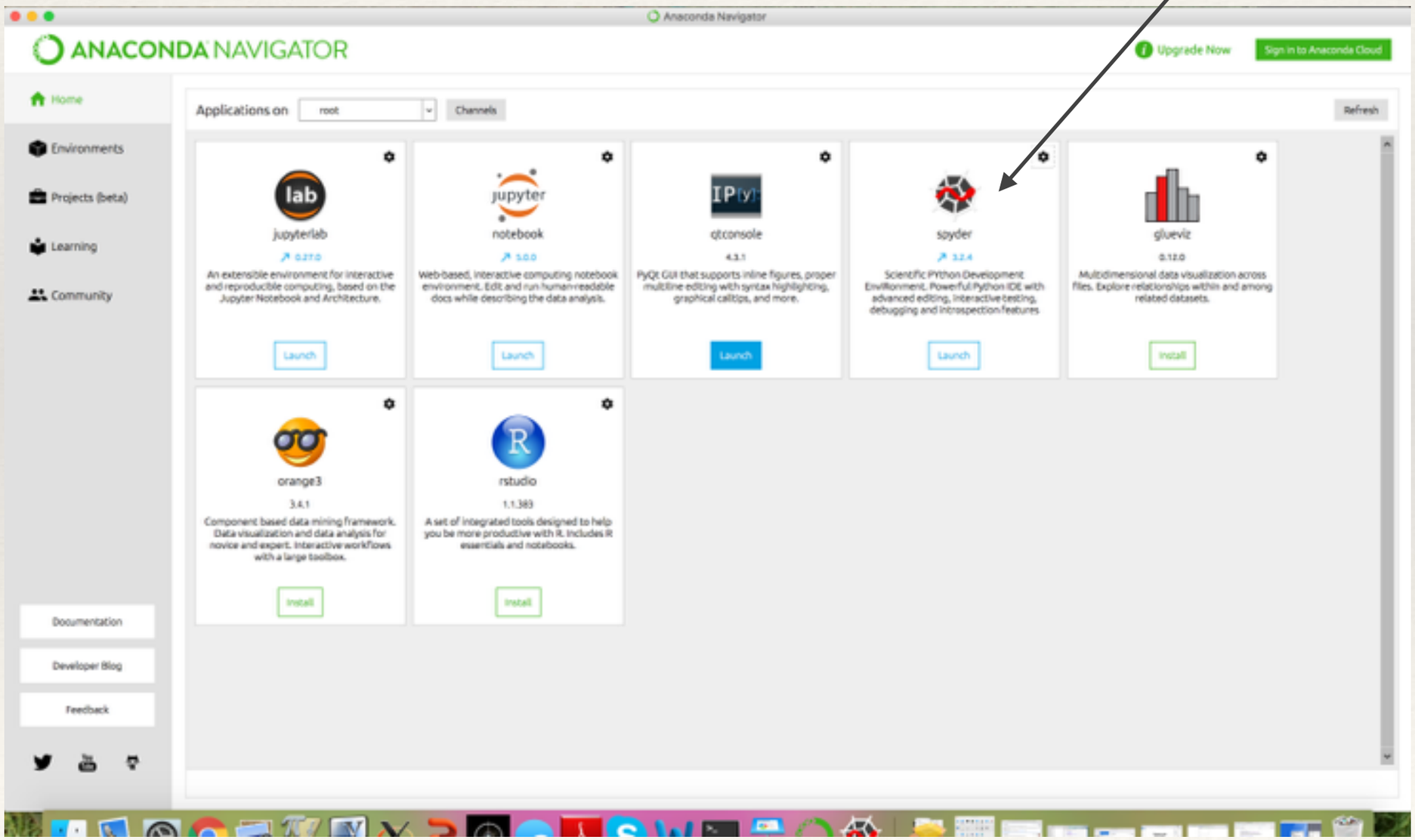
- If you get stuck, ASK GOOGLE
  - Seriously, someone else has probably faced the same problem, stack exchange is amazing.

Just so you know

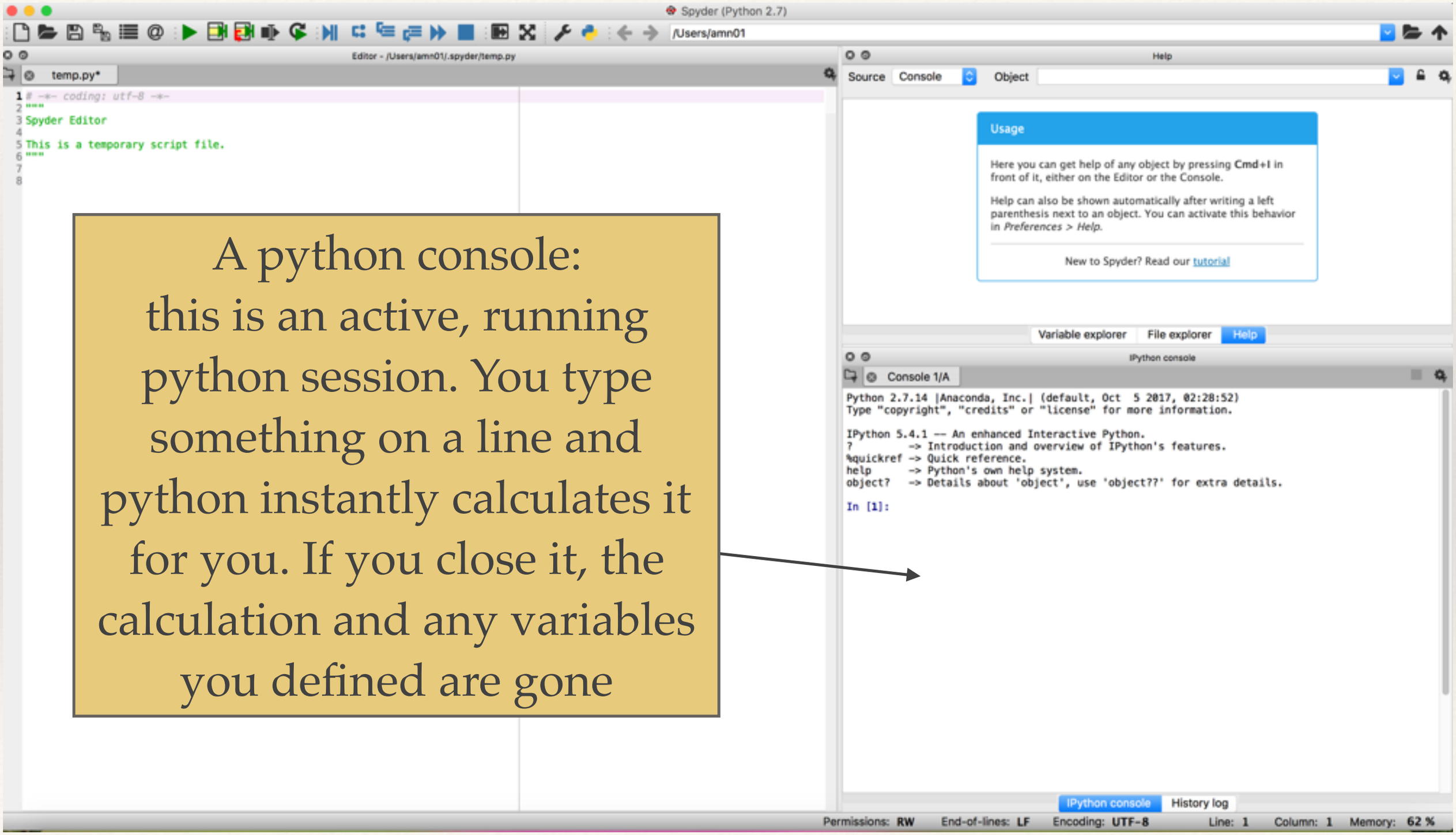
- Python is a ‘non-compiled’ language: this roughly means that your computer translates it into machine code each time you run a program (even if you haven’t made changes). This is slower than e.g. C or Fortran, which you compile first then run. For VERY intensive computations, you do not want to use pure python, however there are ways to combine python with C/Fortran for speed

Lets get started

- open Anaconda Navigator, and select launch spyder

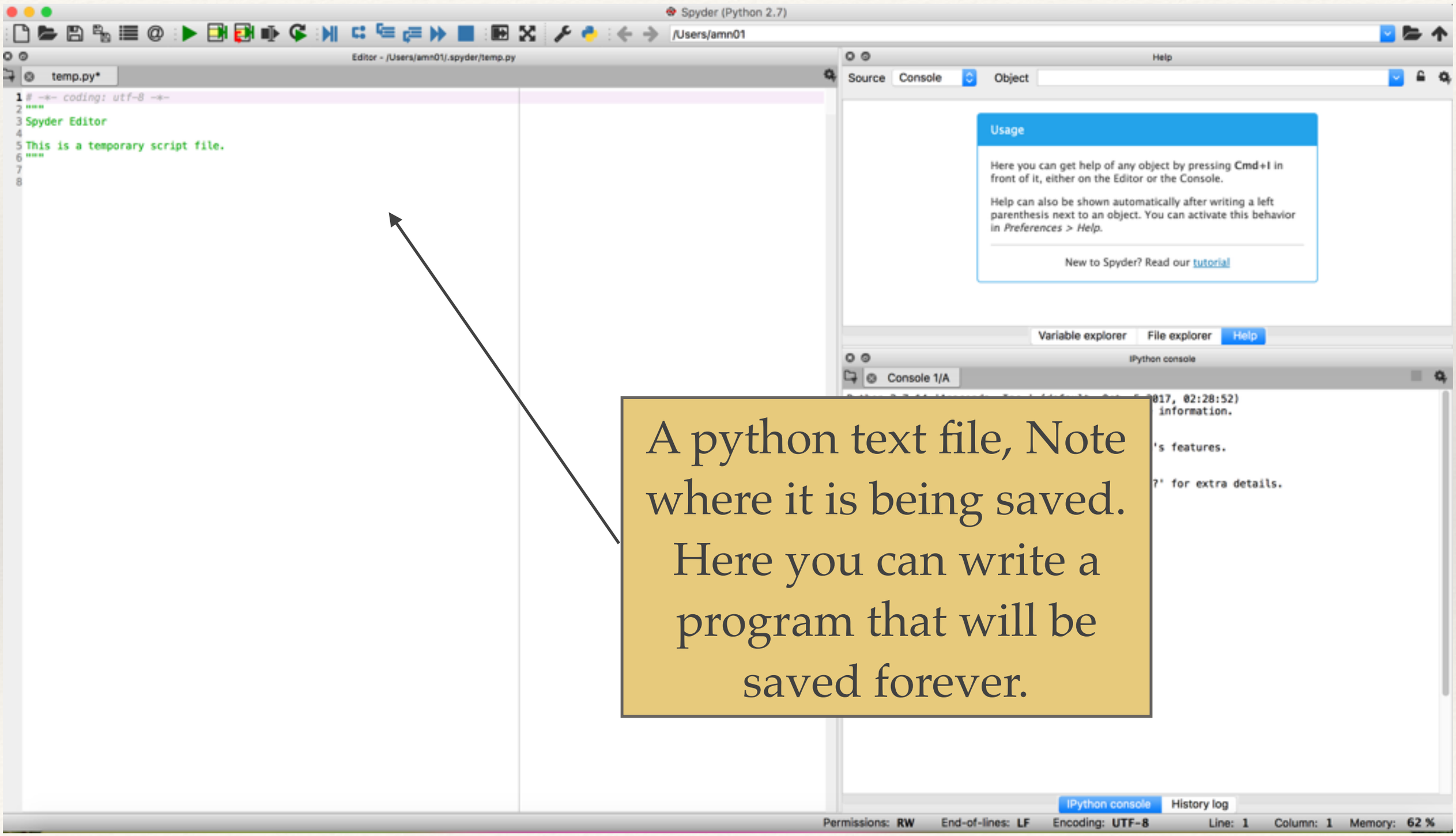


Overview of the spyder screen



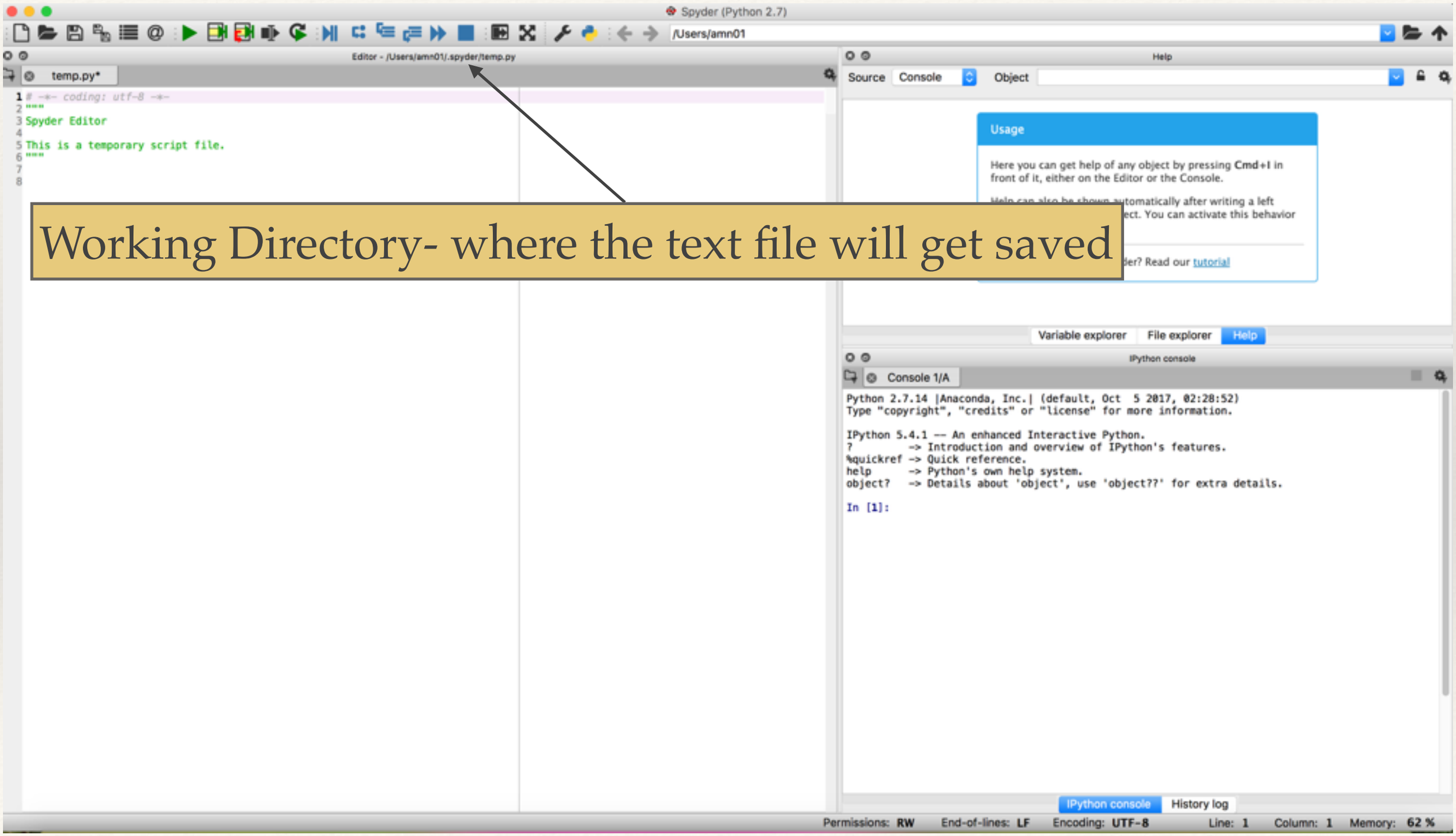
A python console: this is an active, running python session. You type something on a line and python instantly calculates it for you. If you close it, the calculation and any variables you defined are gone

Overview of the spyder screen

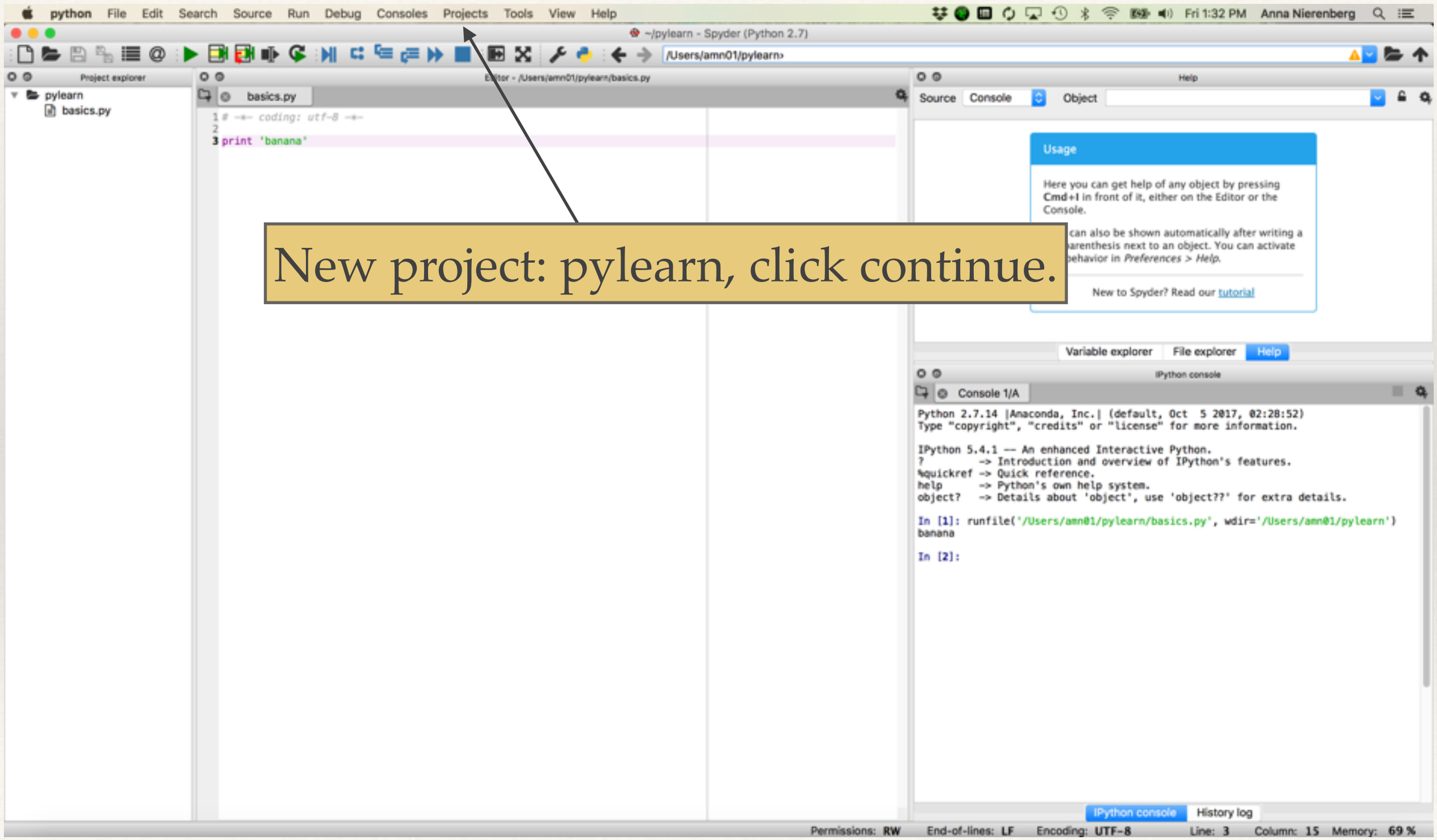


A python text file, Note where it is being saved. Here you can write a program that will be saved forever.

Overview of the spyder screen

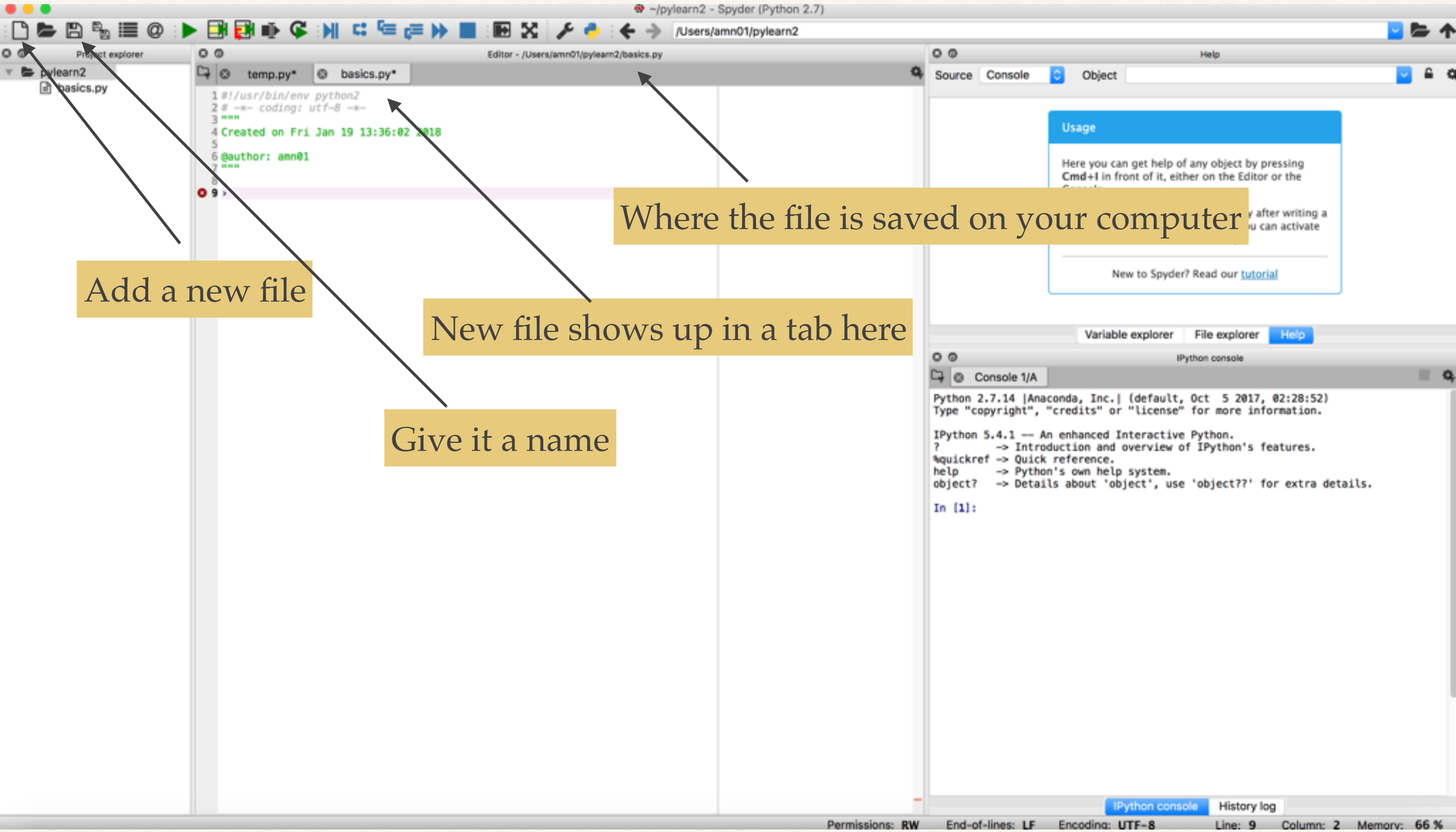


Let's make a new project



New project: pylearn, click continue.

Make a new file in the project



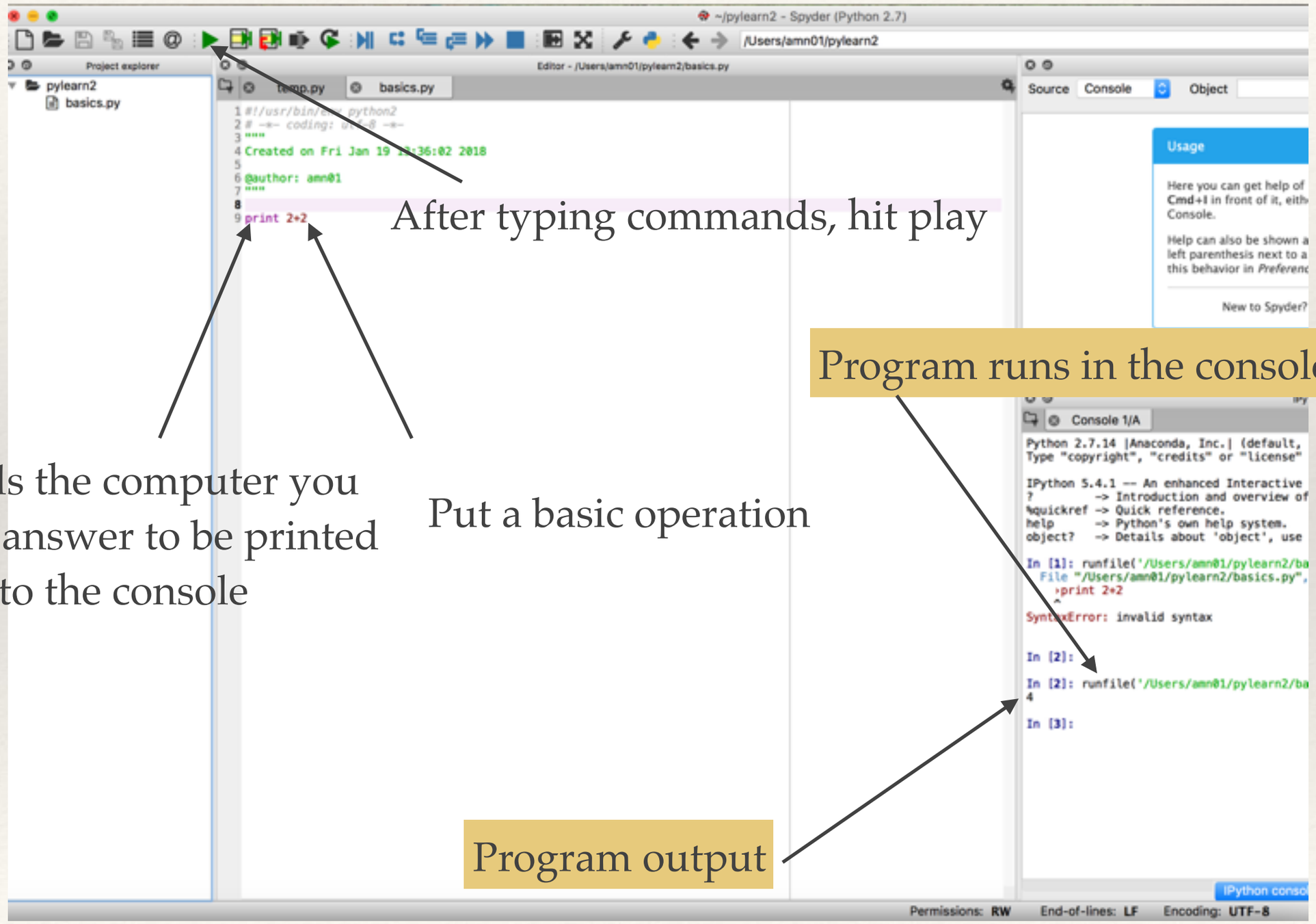
Add a new file

Give it a name

Where the file is saved on your computer

New file shows up in a tab here

Let's write a program



Quick understanding check:

- What happens if you get rid of the print command?

General pro tip

- You do not need all of this to program in python. It is possible to simply run a python program by opening a terminal and typing ‘python program.py’ on the command line. Let’s try that now for those of you with mac/linux. (Maybe Windows but I am not very familiar with the terminal in Windows…)
- The anaconda gui environment is very nice because it’s easy to install

Ways to store information

- assign values to variables
  - x = 5, y = 1
    - print x+y
- see what happens if you try printing a variable you haven't defined

Basic math operations in python

addition	x + y
subtraction	x - y
multiplication	x * y
division	x / y
exponents	x**y

Do some math!

Difference between ‘floats’ and ‘ints’

- Float: Has a decimal point, costs more memory to store
- Int: Integer no decimal point, if every number in an operation is an int then the computer will round the answer to the nearest int value (no float calculations will be made)

What happens if you try running print 2/3?

Pro Tip

- Virtually everyone has wasted time because a number was defined somewhere as an int instead of a float or vice versa

Lists-storing multiple elements

- Make a new list
  favoritenumbers= [51,65,8]
- print favoritenumbers
- List Operations
  to see what the first element is:
  - print favoritenumbers[0]
- to add an element:
  - favoritenumbers.append(22)
- lots of other ‘list’ functions you can look them up.

Square brackets tell python that these numbers are a new list object

ITS A TRAP

If you use python long enough eventually this will cause a bizarre error for you that takes a day to track down…

Understanding Reference Semantics

- **Assignment manipulates references**
  - —x = y **does not make a copy** of the object y references
  - —x = y makes x **reference** the object y references
- **Very useful; but beware!**
- **Example:**

```
>>> a = [1, 2, 3]   # a now references the list [1, 2, 3]
>>> b = a           # b now references what a references
>>> a.append(4)     # this changes the list a references
>>> print b         # if we print what b references,
[1, 2, 3, 4]        # SURPRISE!  It has changed…
```

Why??

b and a are being assigned to the same bits in memory.

Make a list of strings

- enemies=[‘darth vader’, ‘spiders’]
- Strings are words instead of numbers.
- See what happens if you don’t put the quotes

‘For loop’

- a basic programming concept, do something over and over.

```
1 #!/usr/bin/env python2
2 # -*- coding: utf-8 -*-
3 """
4 Created on Fri Jan 19 13:36:02 2018
5
6
7
8
9 myfavoritethings = ['chocolate','dogs','sunshine']
10 i = 0
11 for item in myfavoritethings:
12     print item
13     i = i+1
14     print i
15
```

For each thing in my list,

new variable name for the current element in the sequence

sequence to iterate over

Translation: For each element in the list, do the things that are in the indented block below (print the element value, add 1 to the variable i and print the value of i)

Try

- Add an UNindented line after the loop that prints the variable i, and the variable item

writing files

- let's use python to write a list to a file

```
1 #!/usr/bin/env python2
2 # -*- coding: utf-8 -*-
3 """
4 Created on Fri Jan 19 13:36:02 2018
5
6 @author: amn01
7 """
8
9 myfavoritethings = ['chocolate','dogs','sunshine']
10
11 favFileName = 'myFavoriteThings'
12 f = open(favFileName,'w')
13 f.write(myfavoritethings[0])
14 f.close()
15
16
```

What happens if you try f.write(myfavoritethings)? Why?

writing to files

- Make a for loop that writes all of the elements of myfavoritethings to a file.

```
1  #!/usr/bin/env python2
2  # -*- coding: utf-8 -*-
3  """
4  Created on Fri Jan 19 13:36:02 2018
5
6  @author: amn01
7  """
8
9  myfavoritethings = ['chocolate','dogs','sunshine']
10
11 favFileName = 'myFavoriteThings'
12 f = open(favFileName,'w')
13
14 for item in myfavoritethings:
15         f.write('%s \n'%item)
16
17 f.close()
```

reading files

```
f = open('myFavoriteThings')

flines = f.readlines()

print flines

for line in flines:
    print line
```

- readlines is a file operation that returns a list of strings separated by ‘\n’ characters

Writing numbers to files

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Fri Jan 19 13:36:02 2018

@author: amn01
"""

myfavoriteFloats = [34.6, 38.2, 100.1]

f = open('myfavoritefloats','w')

for flt in myfavoriteFloats:
        f.write('%f \n'%flt)

f.close()



floatFi = open('myfavoritefloats','r')

floatLines = floatFi.readlines()

for line in floatLines:
        print line
        print float(line) +1
```

- What happens if you try print line+1 ?

Yayyyy!!! Good job

These meerkats are really proud of you!

Or maybe they want to eat us? It’s unclear

Interlude: Difference between the file and the console

- The file is saved forever
- Console things disappear when the console is closed. Rerunning them is annoying. console is for quick checks only.

Functions and modules

Functions make it easy to do the same operation for new data

Function Name

Function arguments (can have any number of arguments)

```
def writeFloatListToFile(floatList,fileName):

    f = open(fileName,'w')

    for flt in floatList:
            f.write('%f \n'%flt)

    f.close()
```

```
myfavoriteFloats = [34.6, 38.2, 100.1]

outName = 'floatFile'

writeFloatListToFile(myfavoriteFloats,outName)

myNEWfavoriteFloats = [1., 1., .1]


newOutName = 'betterfloatFile'

writeFloatListToFile(myfavoriteFloats,newOutName)

```

Functions can also return values to you so you can keep calculating

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Fri Jan 19 13:36:02 2018

@author: amn01
"""



def sumOfFloats(floatList):
    floatSum = 0
    for element in floatList:
        floatSum = floatSum+ element


    return floatSum
myfavoriteFloats = [34.6, 38.2, 100.1]

floatSumOut = sumOfFloats(myfavoriteFloats)

print floatSumOut

print floatSumOut+300

```

Functions can be in their own files and you can import them

Editor - /Users/amn01/pylearn2/listFunctions.py

temp.py | listFunctions.py*

```
1  #!/usr/bin/env python2
2  # -*- coding: utf-8 -*-
3  """
4  Created on Fri Jan 19 13:36:02 2018
5
6  @author: amn01
7  """
8
9
10
11 def sumOfFloats(floatList):
12     floatSum = 0
13     for element in floatList:
14         floatSum = floatSum+ element
15
16
17     return floatSum
18
19 def printList(stringList):
20
21     for element in stringList:
22         print element
23
24
25
26
```

A file with a couple of functions

Editor - /Users/amn01/pylearn2/

temp.py | listFunctions.py | basics.py

```
1  #!/usr/bin/env python2
2  # -*- coding: utf-8 -*-
3  """
4  Created on Fri Jan 19 13:36:02 2018
5
6  @author: amn01
7  """
8
9  import listFunctions
10
11 lst = ['alpha','beta','gamma']
12
13 listFunctions.printList(lst)
14
15
16 floatList = [1.,2.,3.]
17
18 floatSum = listFunctions.sumOfFloats(floatList)
19
20 print floatSum
```

Importing the file lets you use any of the functions

To use functions from the imported file you have to write ‘imported filename. function’

You can also rename things

- import ‘listFunctions’ as lF
- or if you don’t want to have to type listFunctions.function, you can write ‘from listFunctions import *’

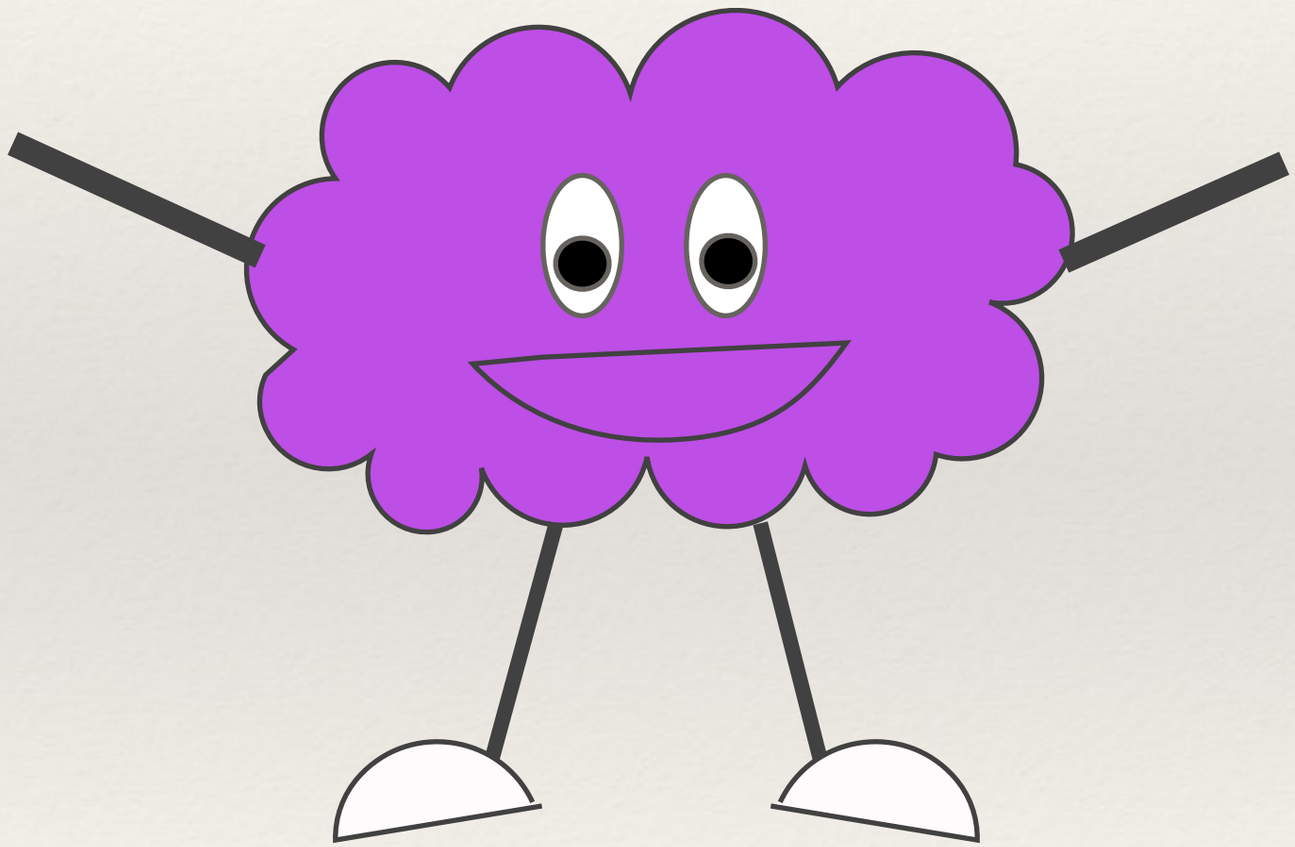
These things can make debugging more challenging, if you forget where you imported a function from. Yes code can really start to get that complicated.

Now it's time for NUMPY

- The workhorse for doing things with numbers in python
- Anaconda comes with the numpy modules so you can just type import numpy in any file to have full access

Side note:

Normal people call it ‘num-pie’. It’s too late for me.



(the numpy monster, artist’s rendition)

The main objects of numpy is arrays

- arr = numpy.array([1,2,3])
  - arrays are different from lists- arrays come with a different set of built in functions
  - e.g. you can type arr.mean(), arr.std() to get the mean and standard deviation of an array there are LOTS more, the internet knows all of them

Array operations

- adding/multiplying, dividing a number to an array adds/multiplies/divides it to every element in the array
  - try numpy.array([1,2,3]) + 5
- numpy.array([1,2,3])**2 gives ([1,4,9]), *squares each element separately* **NOT** a matrix operation. There is a whole separate package for matrix operations (linalg)
- operations bet. multiple arrays are done element by element try arr([1,2,3]) + arr([4,5,6])

Arrays can have multiple dimensions

- arr = numpy.array([[1,2,3],[11,12,13]])
- to specify an array element:
  - element = arr[i,j] the i tells you which element from the zeroth axis and indicates which element from the first axis, here, arr[0,1] would give 2, arr[1,0] would give 11, try to figure out what the index of the element 13 is
- Arrays can have arbitrary numbers of dimensions:
  arr = numpy.array([[[1,2],[3,4]],[[11,12],[13,14]]])
  - arr[0,0,0] = 1

Array indexing continued

- You can pull out an entire column or row using ':'
- arr = numpy.array([[1,2,3],[11,12,13]])

arr[:,0] means give me the zeroth element of everything along the first axis (==>gives the first column)

arr[:,0] = numpy.array([1,11])

More array indexing

arr = numpy.array([[[1,2],[3,4]],[[11,12],[13,14]]])

- arr[0,:,:] = numpy.array([[1,2],[3,4]])
- try arr[:,0,:]

check, how do you pull out element 14

make a sequential array!

- Try out:
  - arr = numpy.linspace(1,100,2)
  - arr = numpy.arange(1,100,2)

- I still cannot remember which one is which…

Use numpy to save arrays to files

- Only works with arrays that have less than or equal to two dimensions

numpy.savetxt(outputFileName, array)

laziness

- If you want to be lazy,
- arr = numpy.array([[[1,2],[3,4]],[[11,12],[13,14]]])
- arr[0] = arr[0,:,:,… (however many dimensions)], automatically pulls out the first element of the zeroth axis., in the above case returns numpy.array([[1,2],[3,4]])

This makes debugging and understanding people's code difficult when things get complicated because you have to remember how many dimensions the array had in the first place. I definitely use this sometimes though.

Use numpy to read numerical files

- arr = numpy.loadtxt(filename)
  - automatically ignores lines with #

  - loadtxt has options to let you ignore certain lines, columns and symbols, and to assume e.g. commas rather than spaces separate numbers

  - loadtxt will be sad if the file is not rectangular (e.g. one row is missing an element).

Lots of random number generators

- Impress your friends!
- gaussianNumbers = numpy.random.normal(mean, standardDeviation, size)
- print gaussianNumbers.mean() print gaussianNumbers.std()