Homework 5

also posted on Brightspace

due next Wed (Oct 12) in class

Homework 5 will be accepted without any late penalty so long as it is turned in before Thu Oct 13 @11:59pm

`Homework5.pdf` (written description)

`Homework5.ipynb`

`BoysBW.jpg`

Homework 6

Homework 6 will be posted this week (maybe Wed), and it will be due in two week, on Wed Oct 26

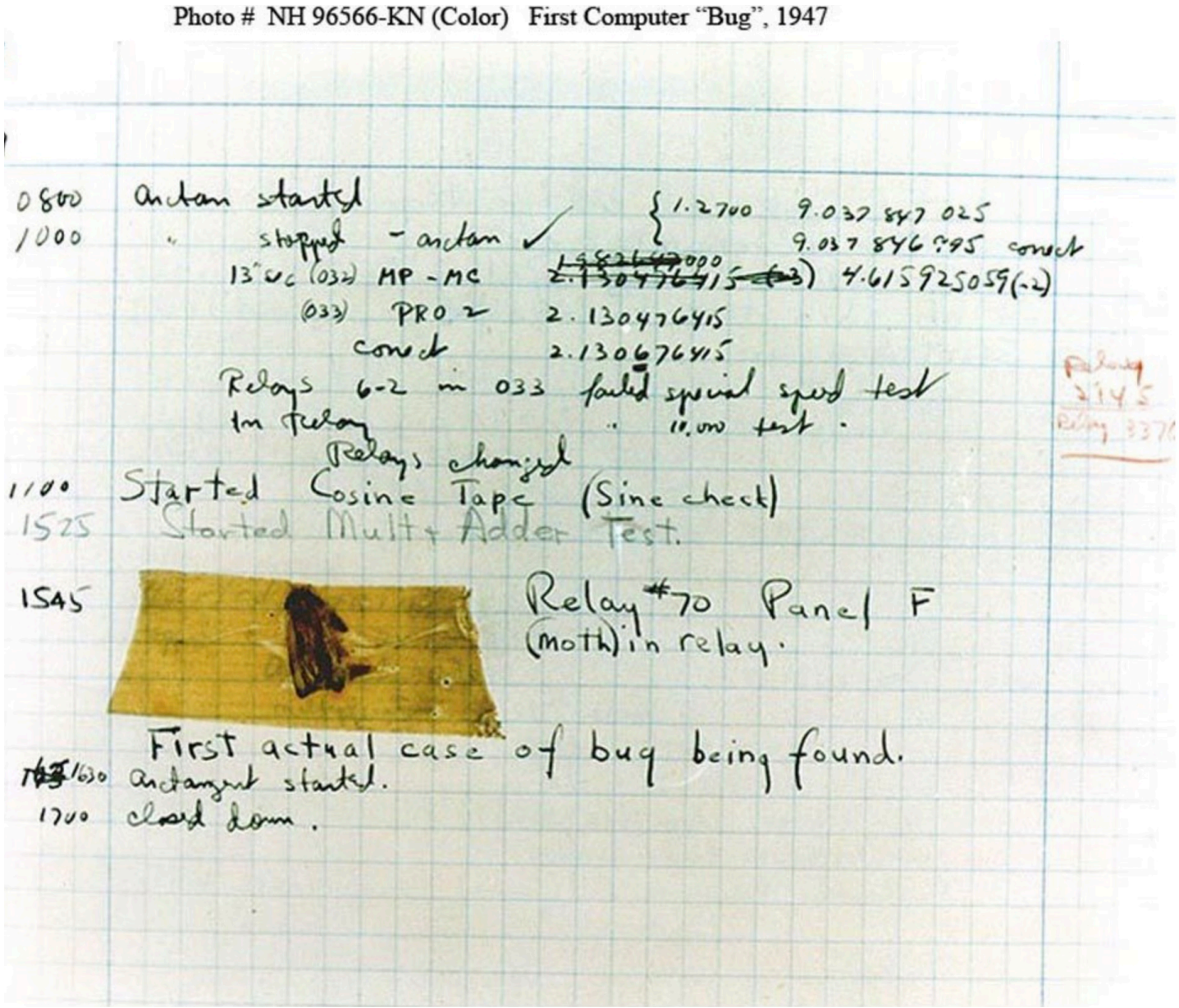
(of course, you do not need to work on it over fall break)

it will be a little bit longer since you have a few more regular class days to work on it

download from Brightspace

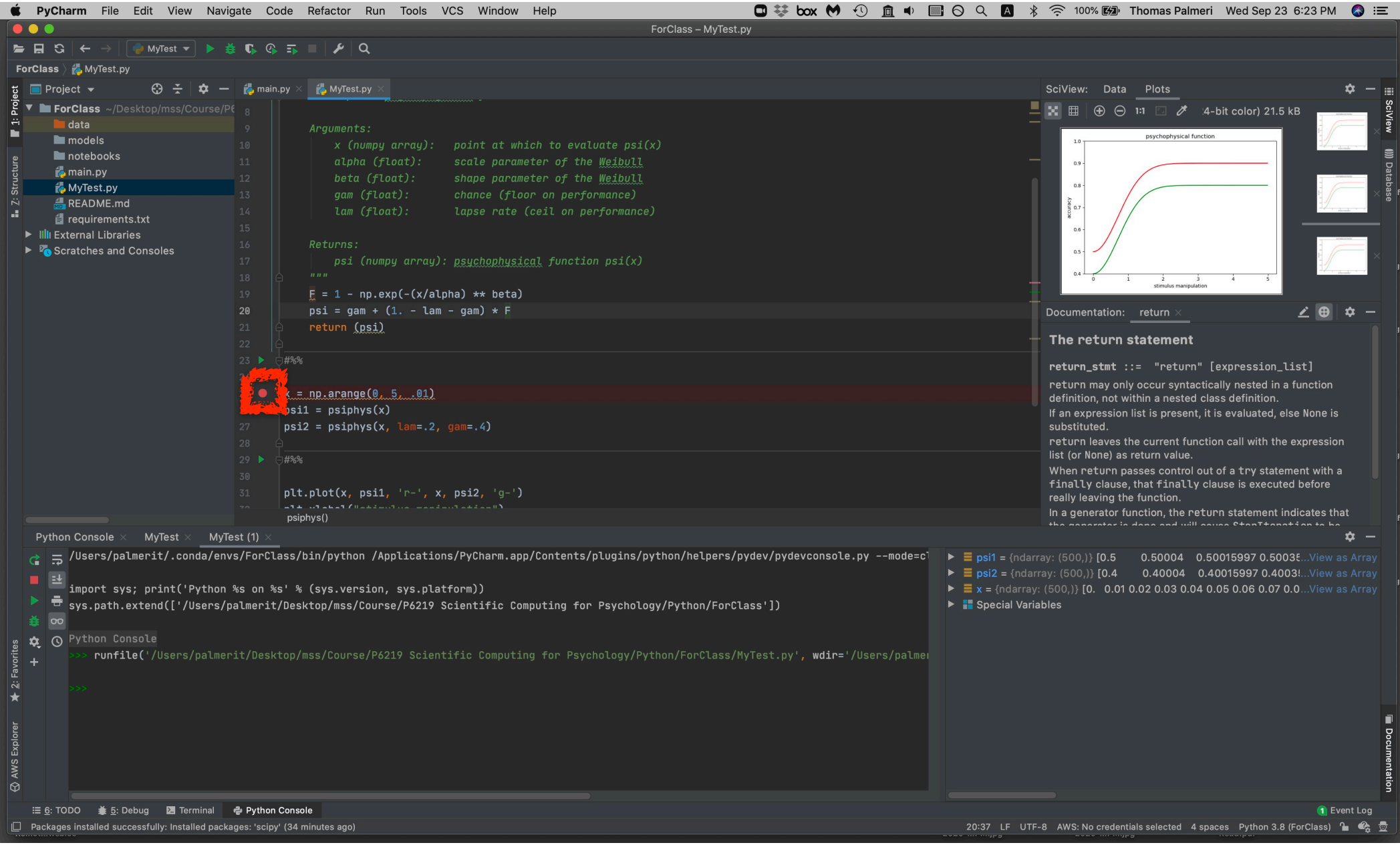
`Week7.zip` (contains .py files)
`Random.ipynb`

debugging in PyCharm

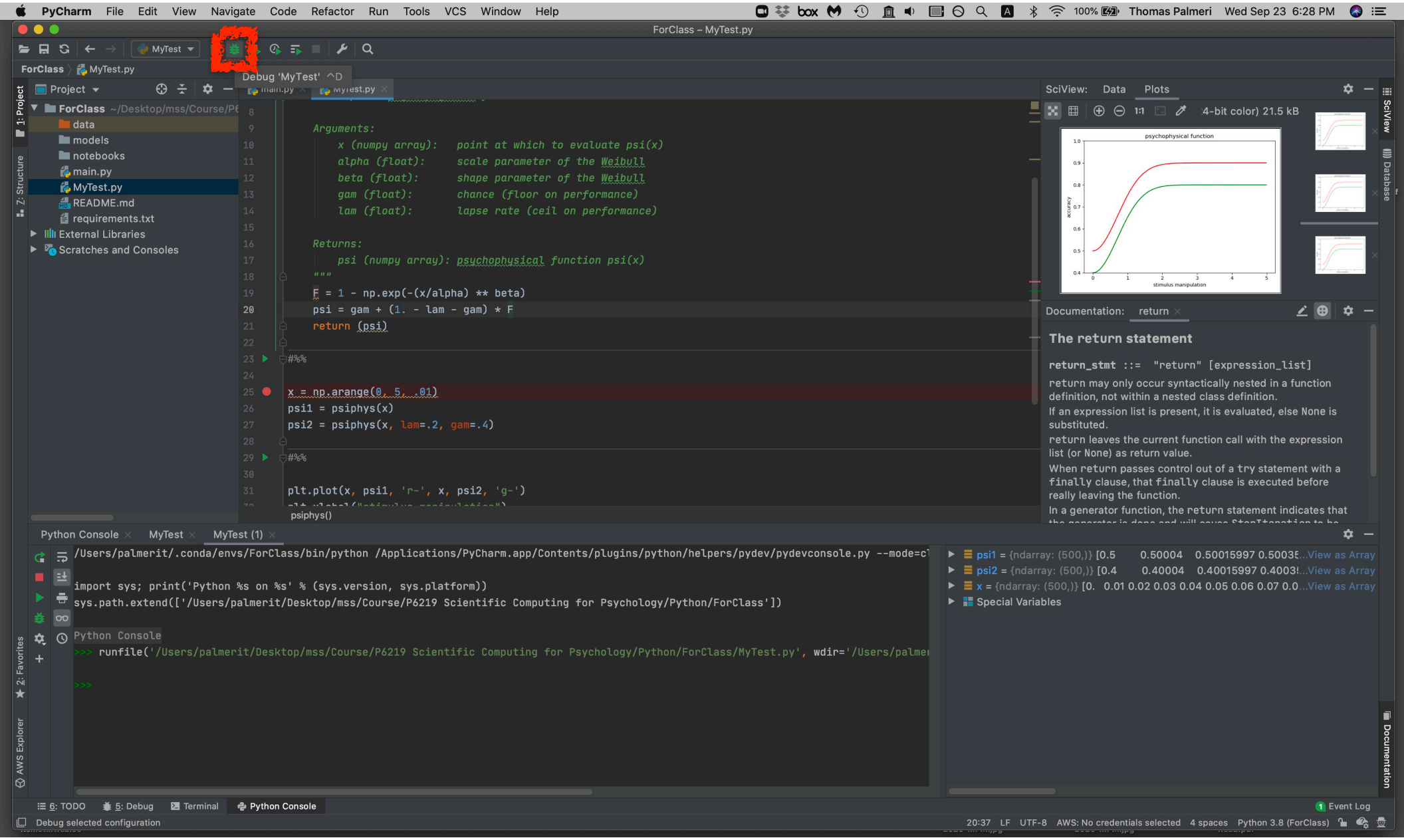


Edison actually coined the term bug (in a system) before computers were around

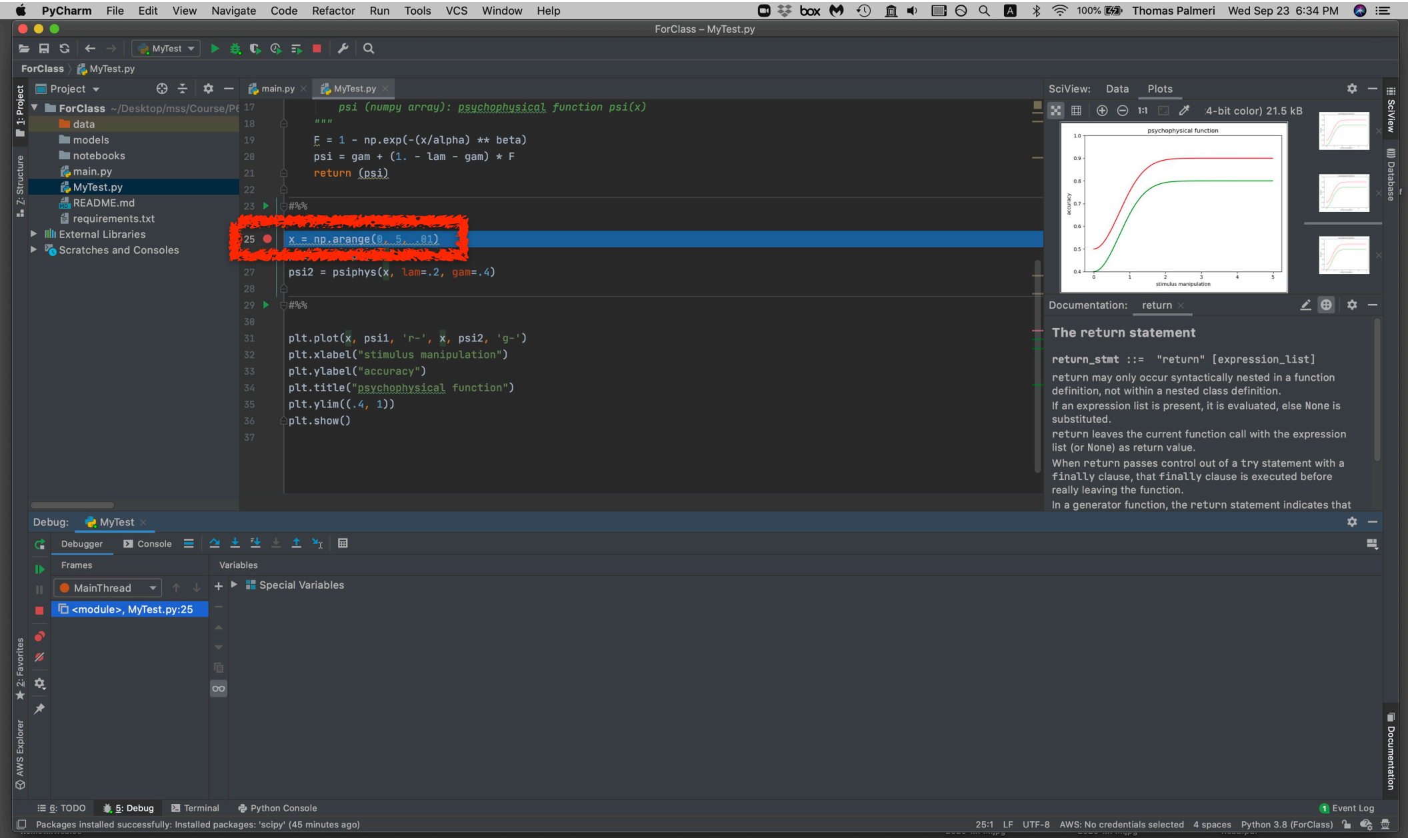
click in the "gutter" (in same column as green arrows) to set a **breakpoint** (can have many)

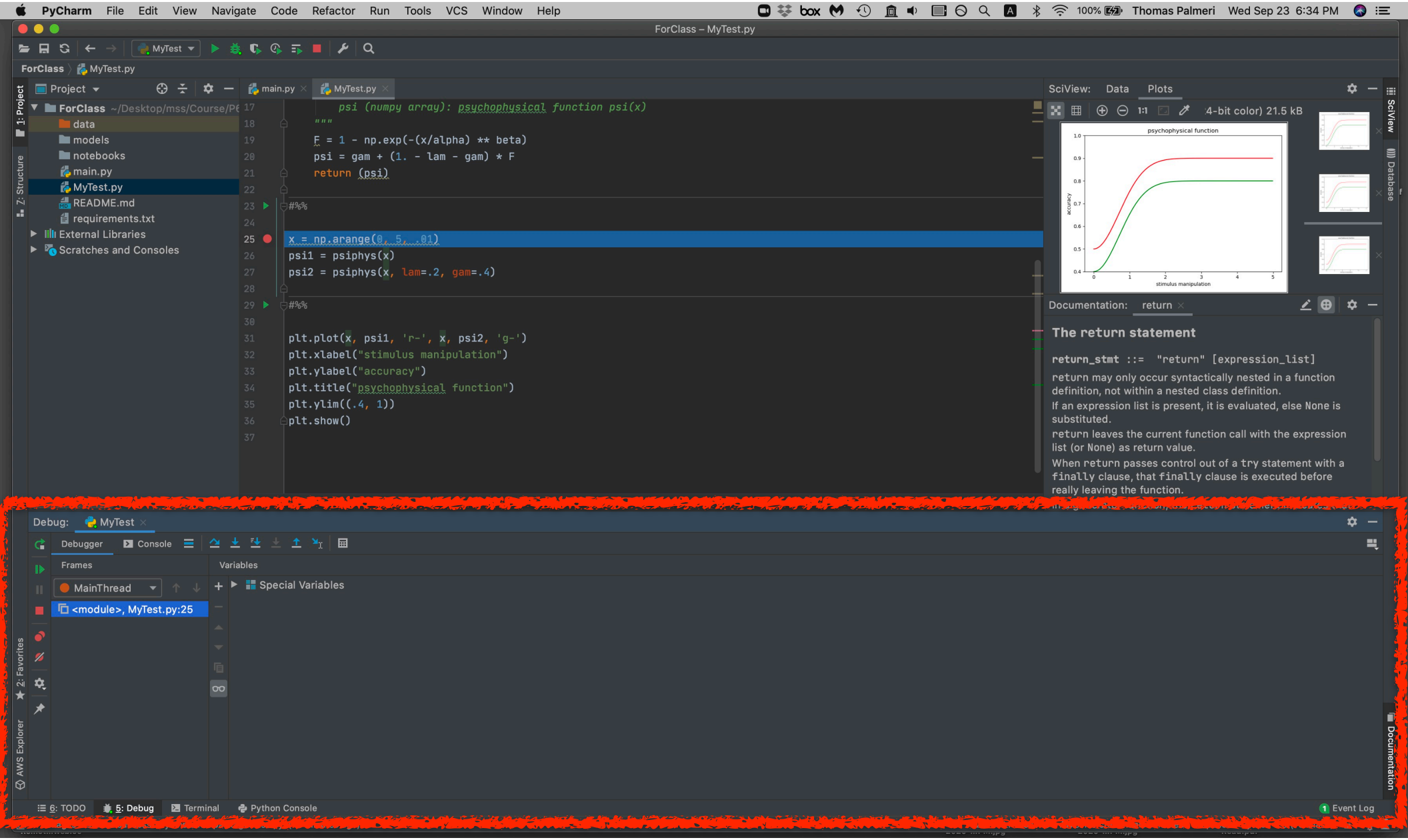


click on that to debug (it's supposed to look like a bug, but with my old eyes looks like a turtle)

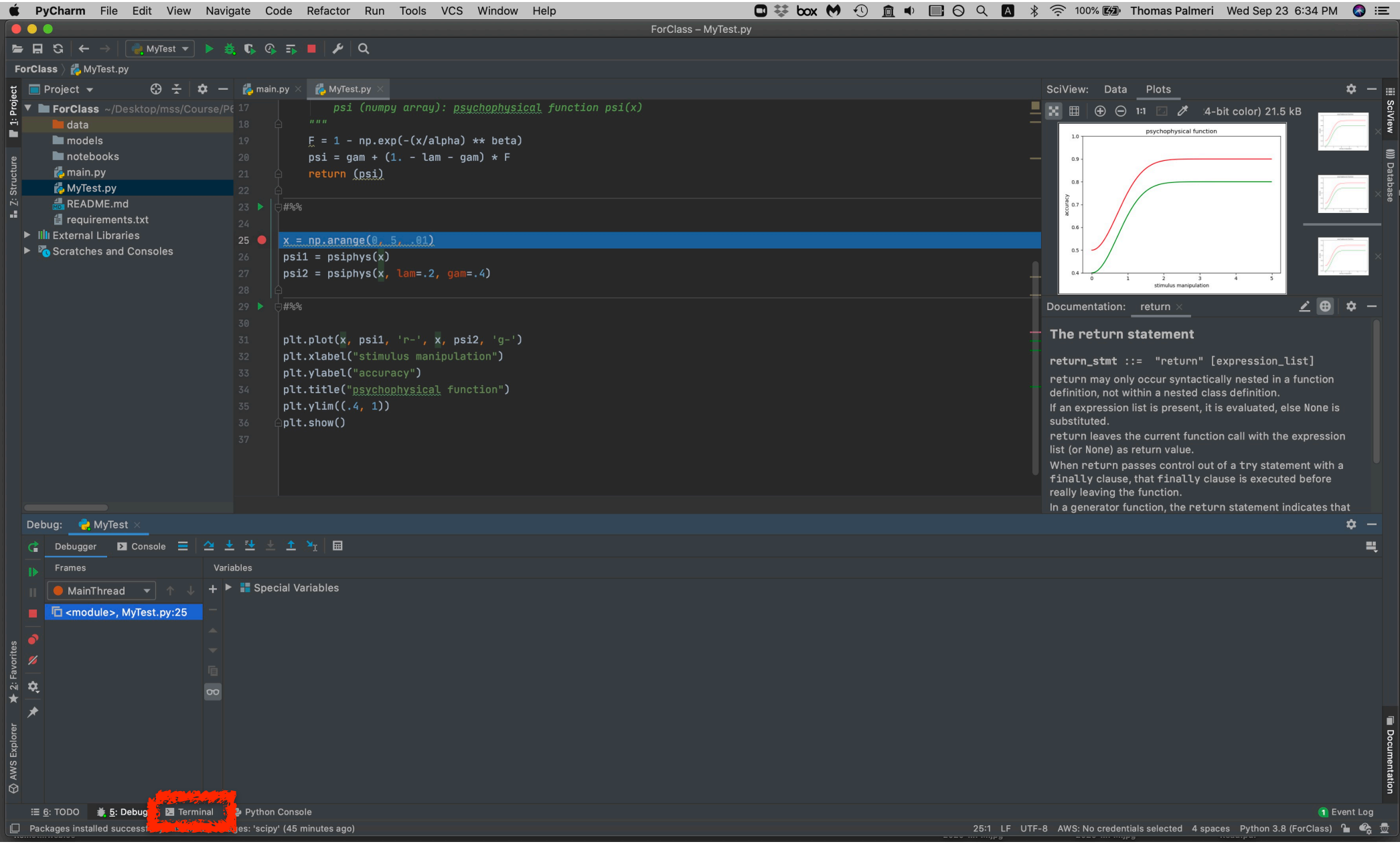


code suspended at that line, but has not executed that line yet

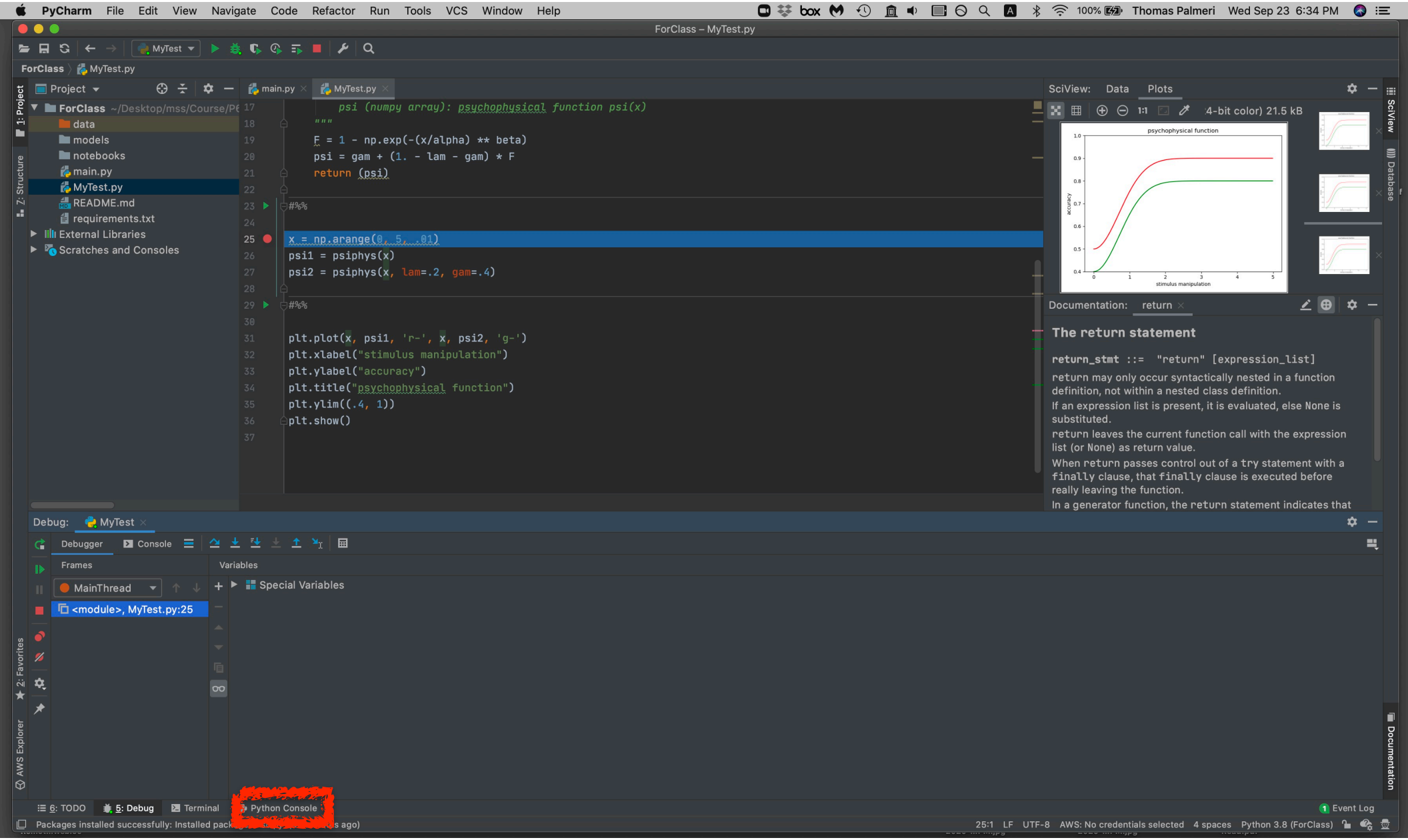




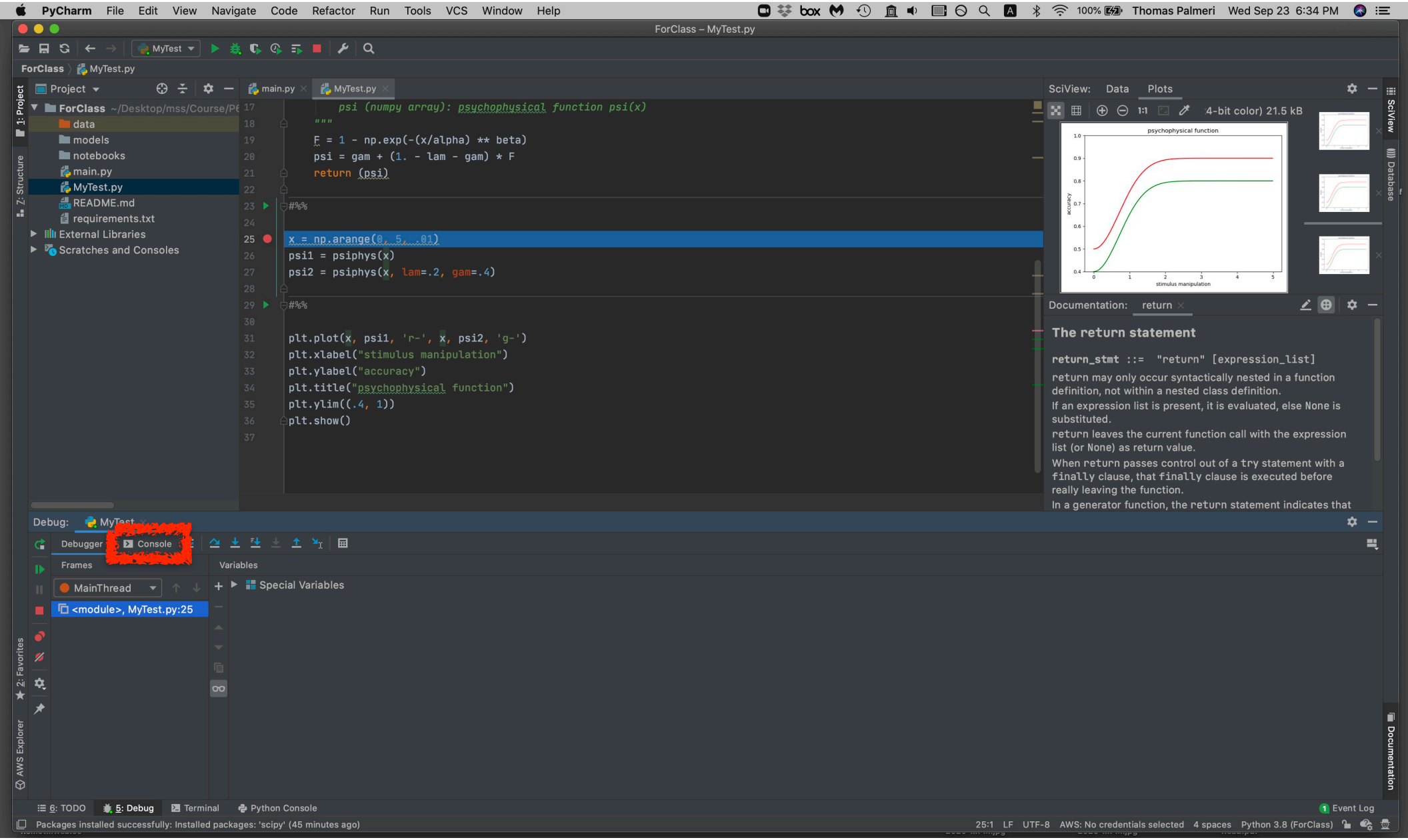
notice that the bottom has switched into Debug mode



can still access the Terminal



and the independent Python Console



the Console tied to MyTest.py is here

PyCharm File Edit View Navigate Code Refactor Run Tools VCS Window Help — Thomas Palmeri Wed Sep 23 6:37 PM

ForClass – MyTest.py

ForClass 〉 MyTest.py

Project

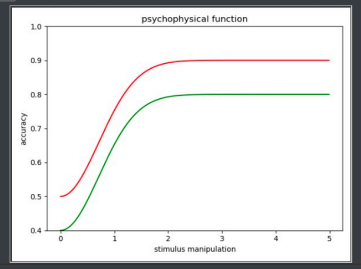
- ForClass ~/Desktop/mss/Course/P6
  - data
  - models
  - notebooks
  - main.py
  - MyTest.py
  - README.md
  - requirements.txt
- External Libraries
- Scratches and Consoles

main.py | MyTest.py

```
19      F = 1 - np.exp(-(x/alpha) ** beta)
20      psi = gam + (1. - lam - gam) * F
21      return (psi)
22
23  #%%
24
25  x = np.arange(0, 5, .01)
26  psi1 = psiphys(x)
27  psi2 = psiphys(x, lam=.2, gam=.4)
28
29  #%%
30
31  plt.plot(x, psi1, 'r-', x, psi2, 'g-')
32  plt.xlabel("stimulus manipulation")
33  plt.ylabel("accuracy")
34  plt.title("psychophysical function")
35  plt.ylim((.4, 1))
36  plt.show()
37
```

SciView: Data Plots

4-bit color) 21.5 kB



Documentation: return

The return statement

```
return_stmt ::=  "return" [expression_list]
```

return may only occur syntactically nested in a function definition, not within a nested class definition.

If an expression list is present, it is evaluated, else None is substituted.

return leaves the current function call with the expression list (or None) as return value.

When return passes control out of a try statement with a finally clause, that finally clause is executed before really leaving the function.

In a generator function, the return statement indicates that

Debug: MyTest

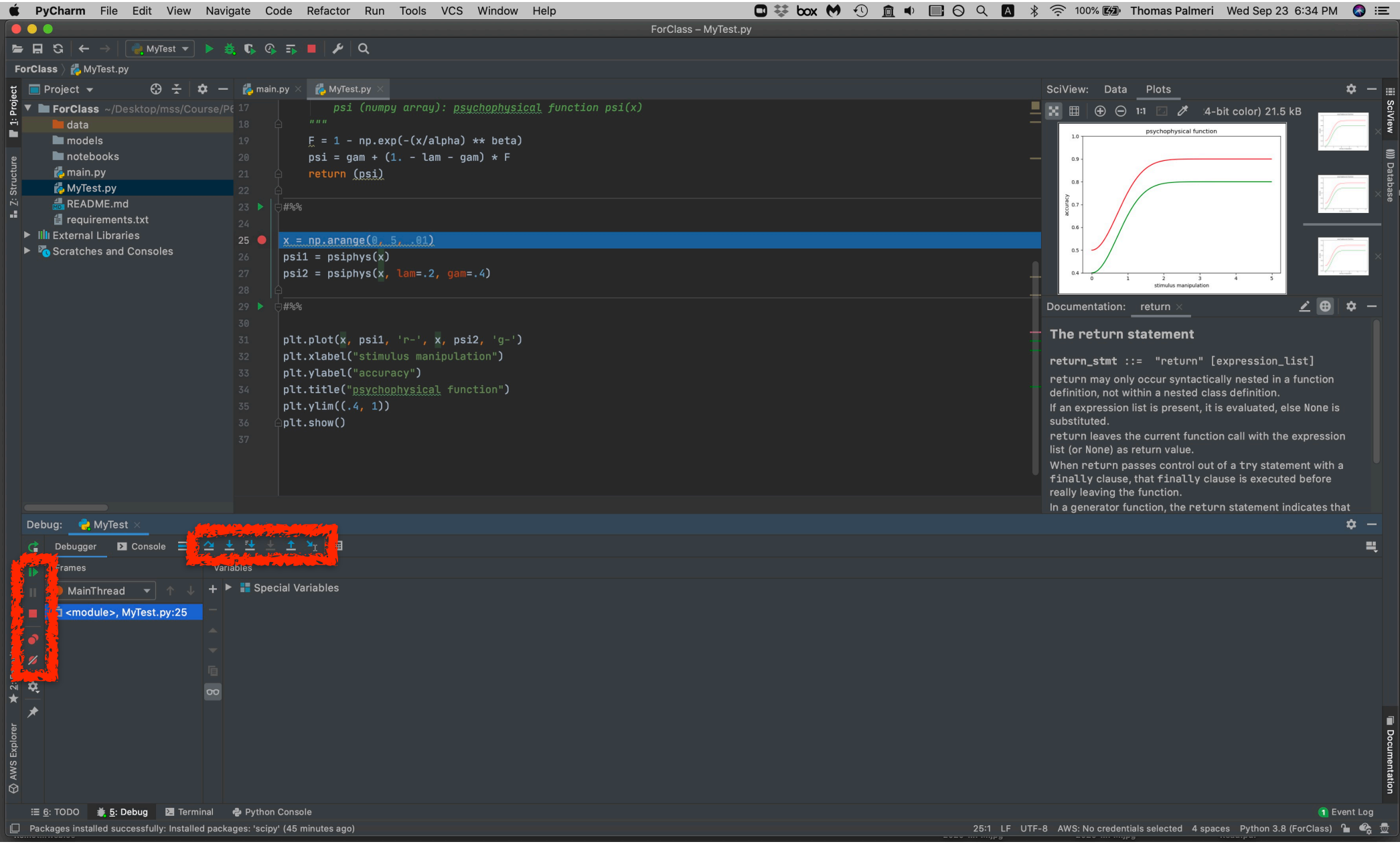
Debugger | Console

```
/Users/palmerit/.conda/envs/ForClass/bin/python /Applications/PyCharm.app/Contents/plugins/python/helpers/pydev/pydevd.py --cmd-line --multiproc --qt-support=auto --client 127.0.0.1 --port 65159 --file "/Us
pydev debugger: process 85316 is connecting

Connected to pydev debugger (build 201.8743.11)

>>>
```

6: TODO | 5: Debug | Terminal | Python Console | Event Log

Packages installed successfully: Installed packages: 'scipy' (48 minutes ago)

5:1 LF UTF-8 AWS: No credentials selected 4 spaces Python 3.8 (ForClass)



these are core debug tools (for stepping through the code)

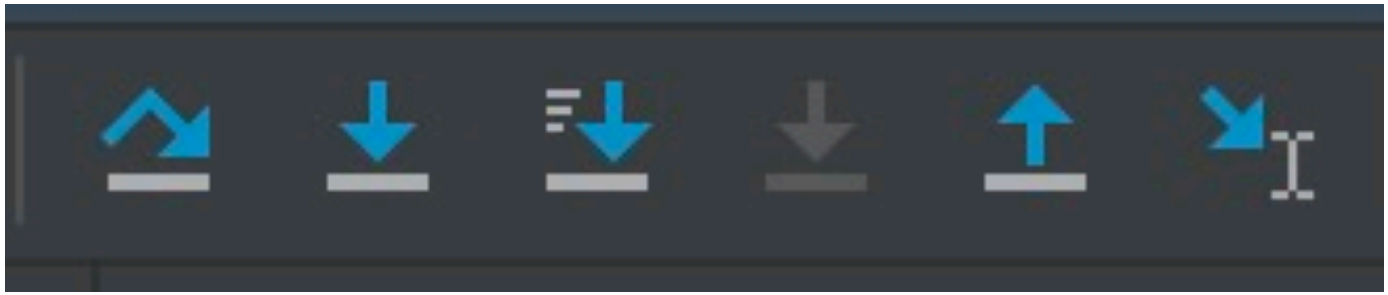
step over

step into

step into *my code*

step out

run to cursor



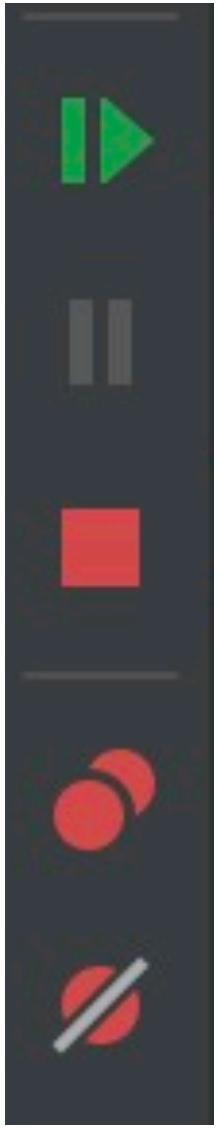
resume program

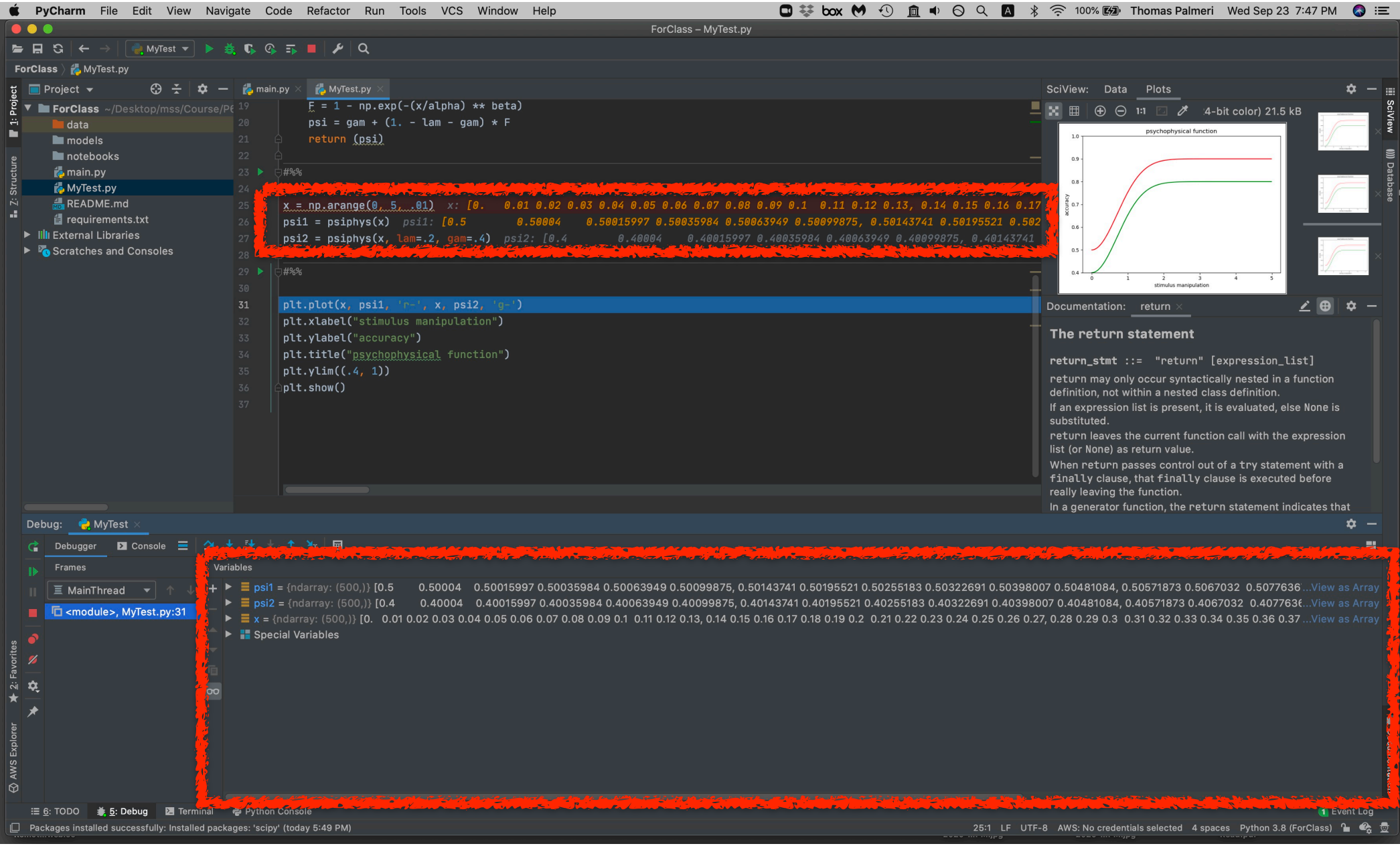
pause program

stop program

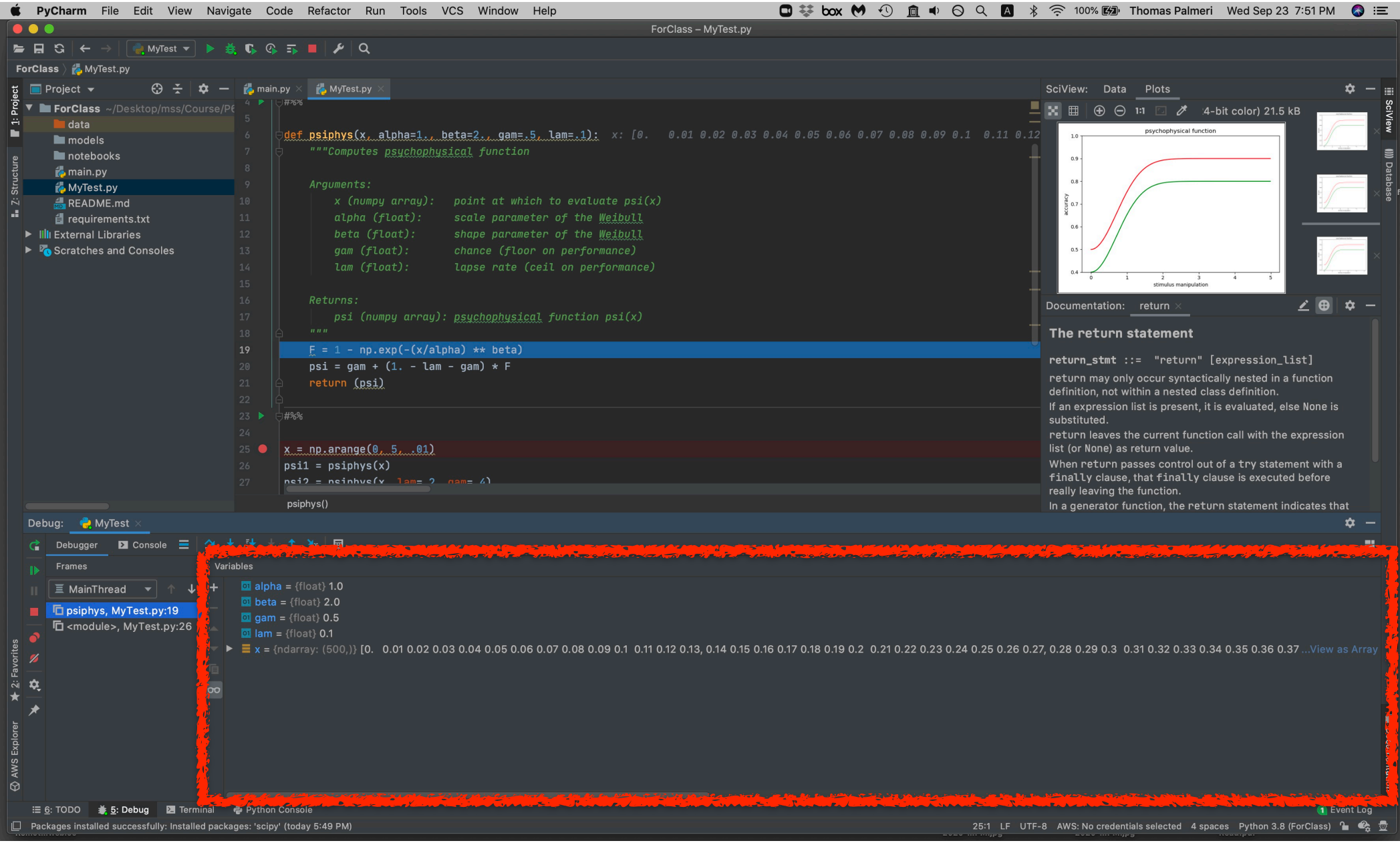
view breakpoints

mute breakpoints

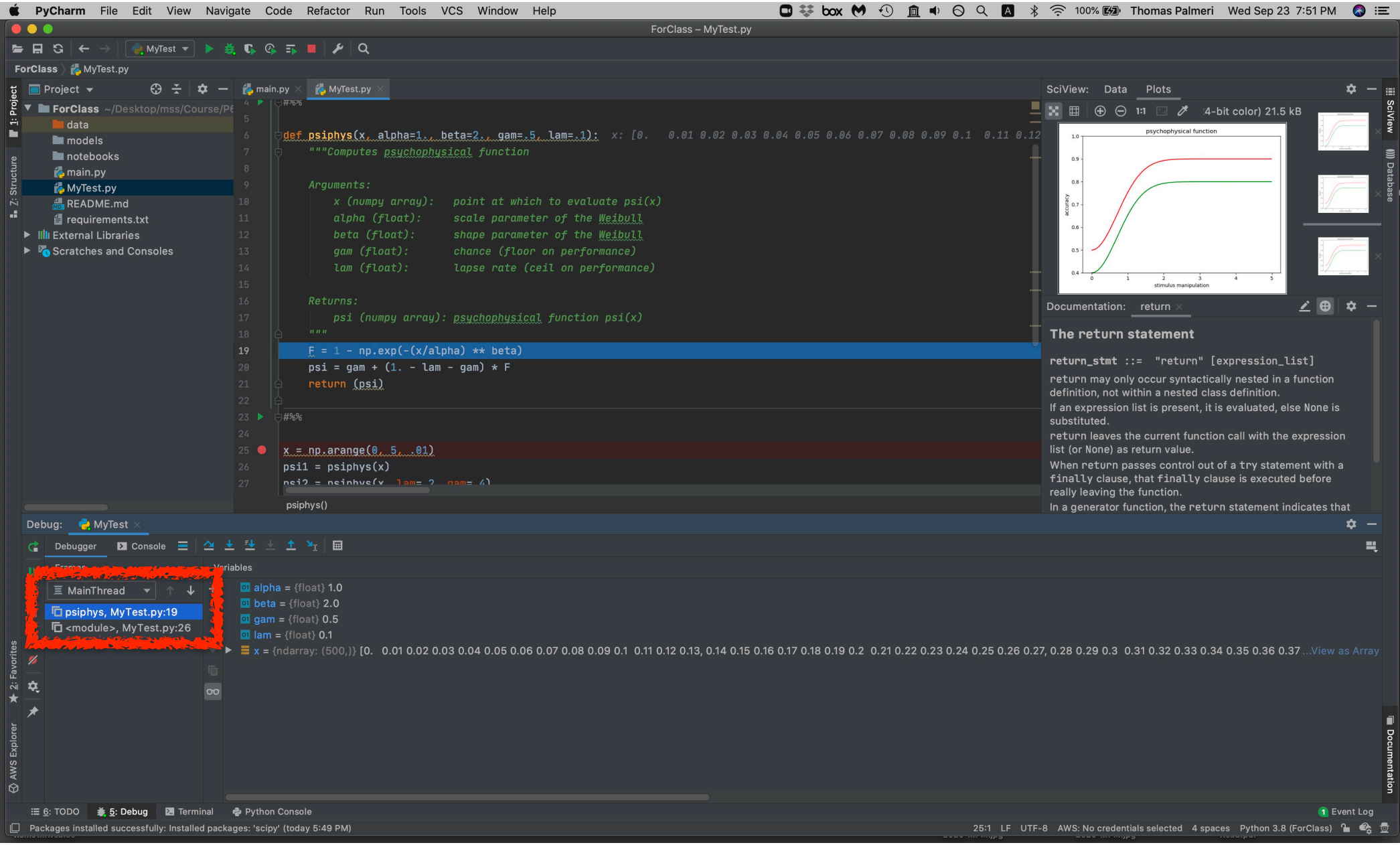




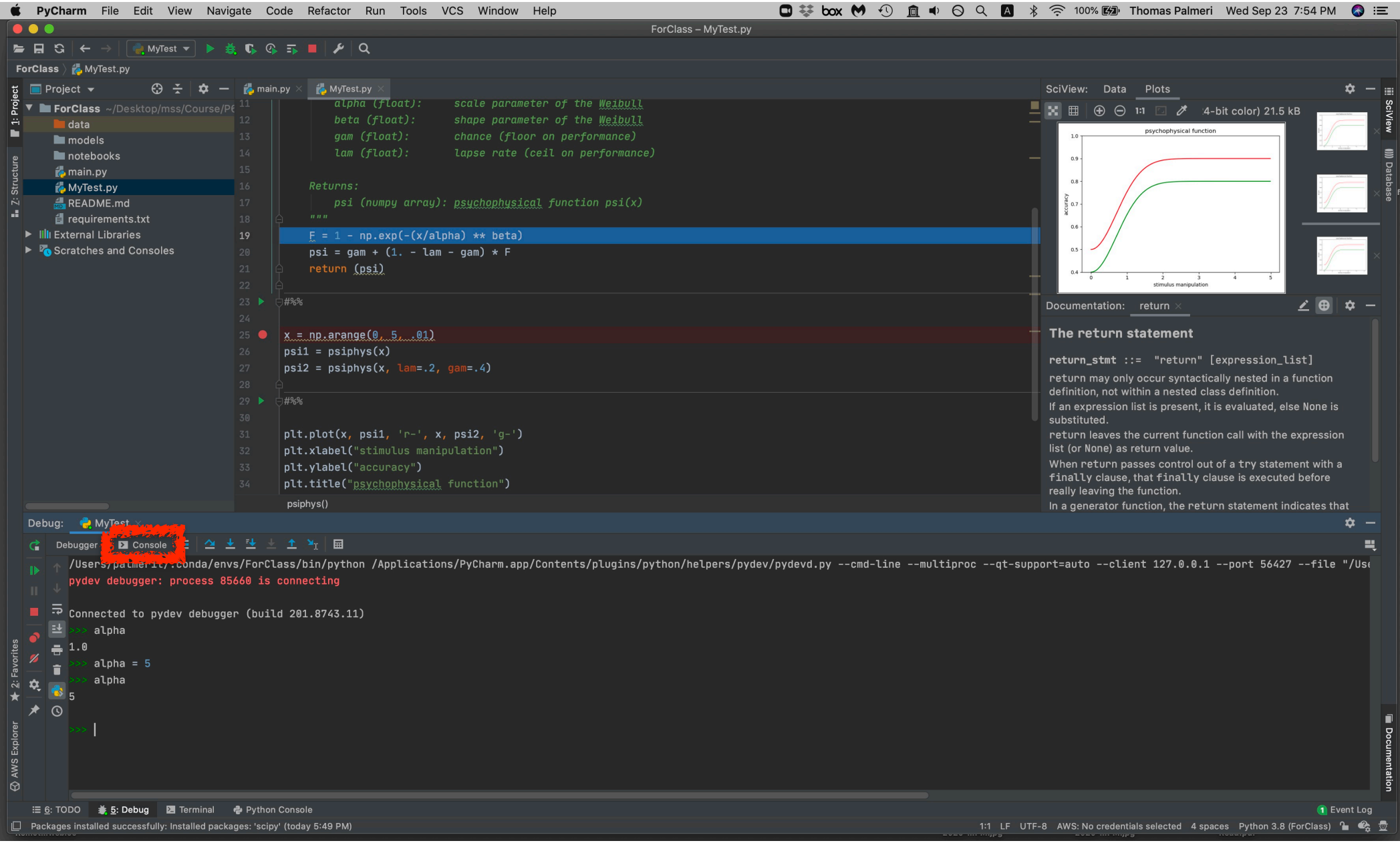
variable values are shown in the code and in the Variables window



stepping into a function, you see the local variables



you can move through the "stack", to see variables outside the function

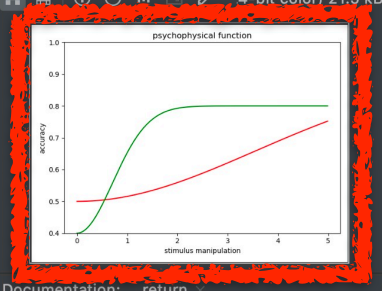


can view and change values of variables in the Console (tied to MyTest.py)

PyCharm File Edit View Navigate Code Refactor Run Tools VCS Window Help — Thomas Palmeri Wed Sep 23 7:55 PM

ForClass – MyTest.py

```
        alpha (float):      scale parameter of the Weibull
        beta (float):       shape parameter of the Weibull
        gam (float):        chance (floor on performance)
        lam (float):        lapse rate (ceil on performance)

    Returns:
        psi (numpy array): psychophysical function psi(x)
    """
    F = 1 - np.exp(-(x/alpha) ** beta)
    psi = gam + (1. - lam - gam) * F
    return (psi)

#%%

x = np.arange(0, 5, .01)
psi1 = psiphys(x)
psi2 = psiphys(x, lam=.2, gam=.4)

#%%

plt.plot(x, psi1, 'r-', x, psi2, 'g-')
plt.xlabel("stimulus manipulation")
plt.ylabel("accuracy")
plt.title("psychophysical function")
```



The return statement

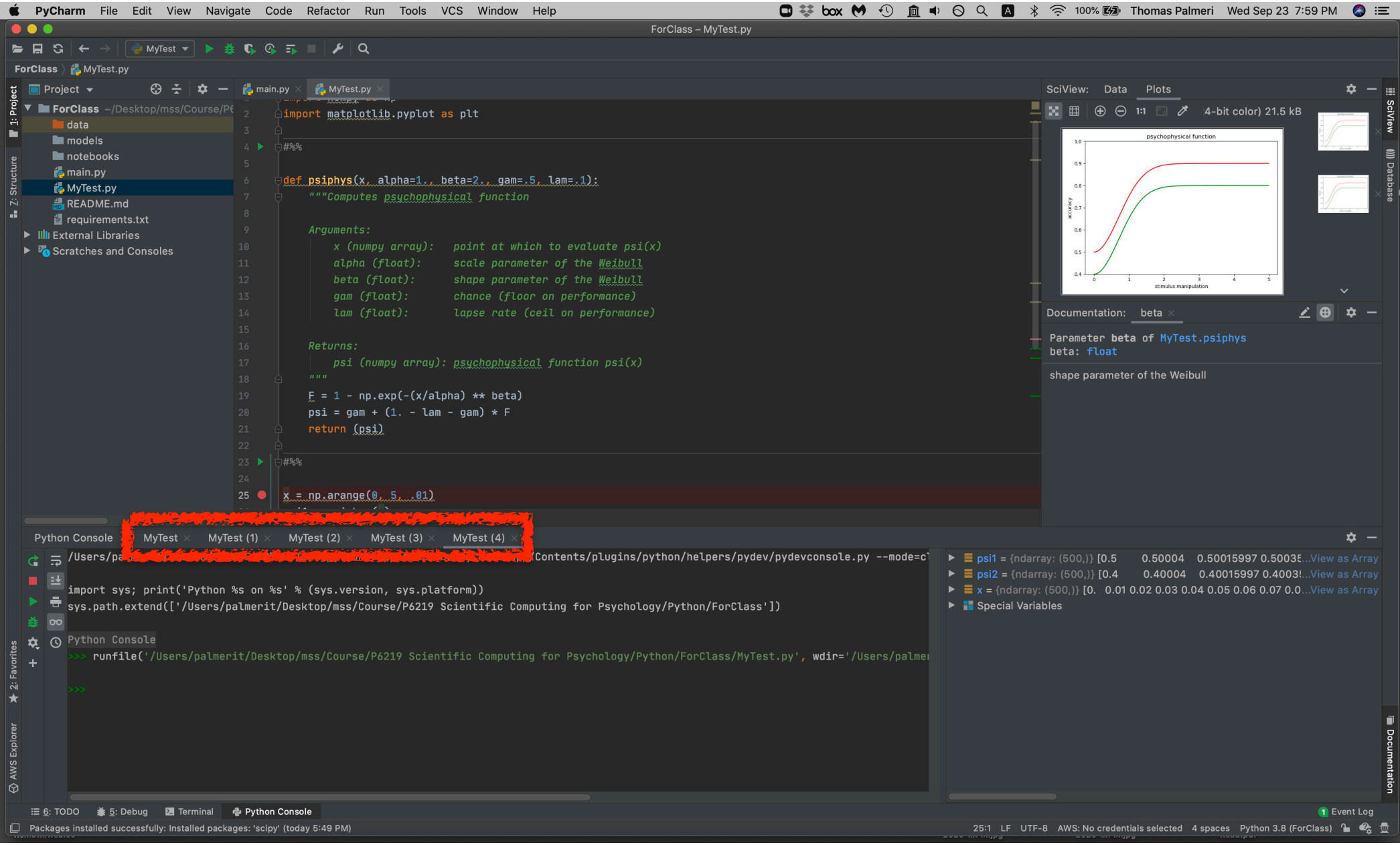
`return_stmt ::= "return" [expression_list]`

return may only occur syntactically nested in a function definition, not within a nested class definition.
If an expression list is present, it is evaluated, else None is substituted.
return leaves the current function call with the expression list (or None) as return value.
When return passes control out of a try statement with a finally clause, that finally clause is executed before really leaving the function.
In a generator function, the return statement indicates that

Frames are not available

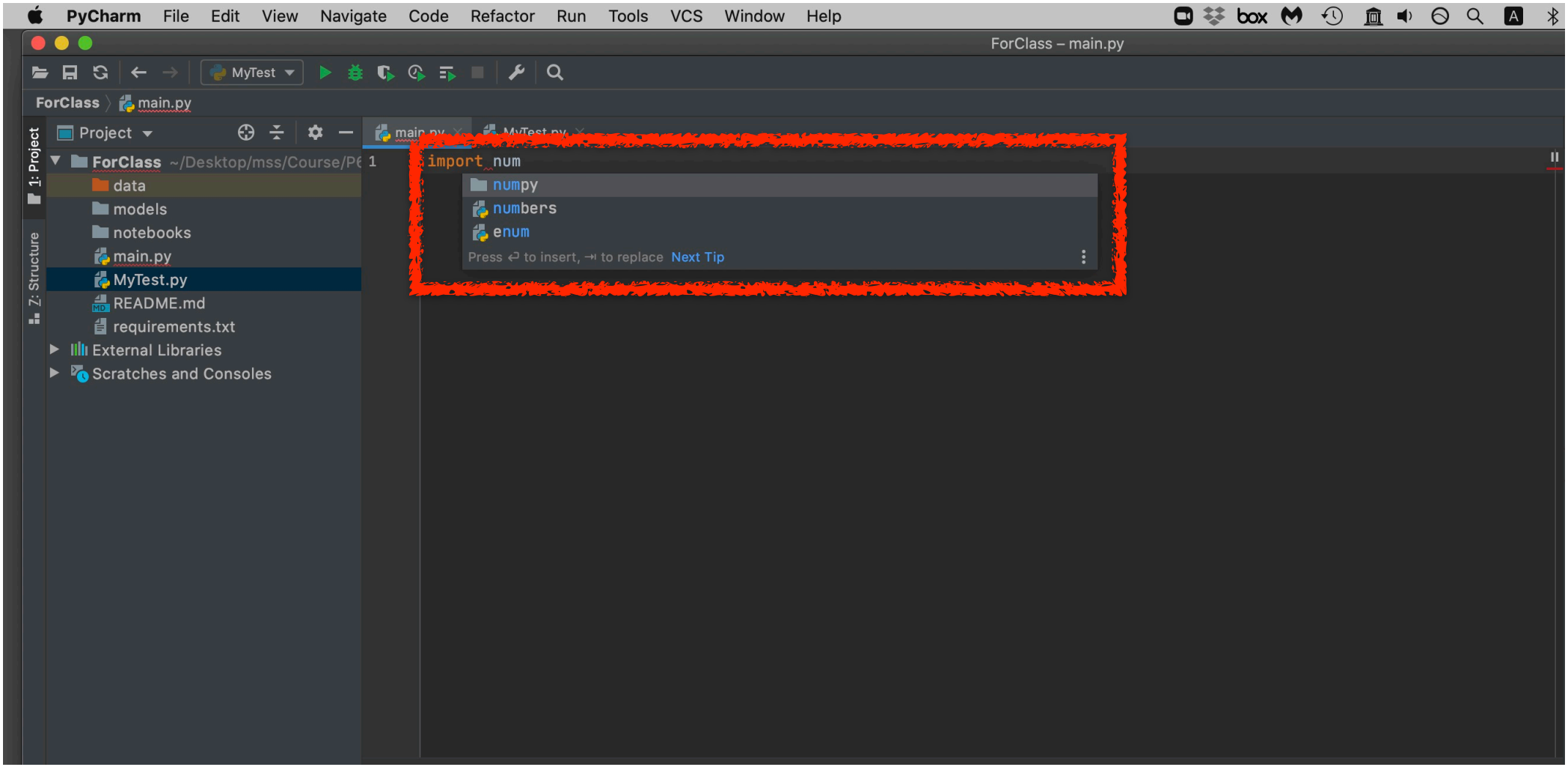
Connected

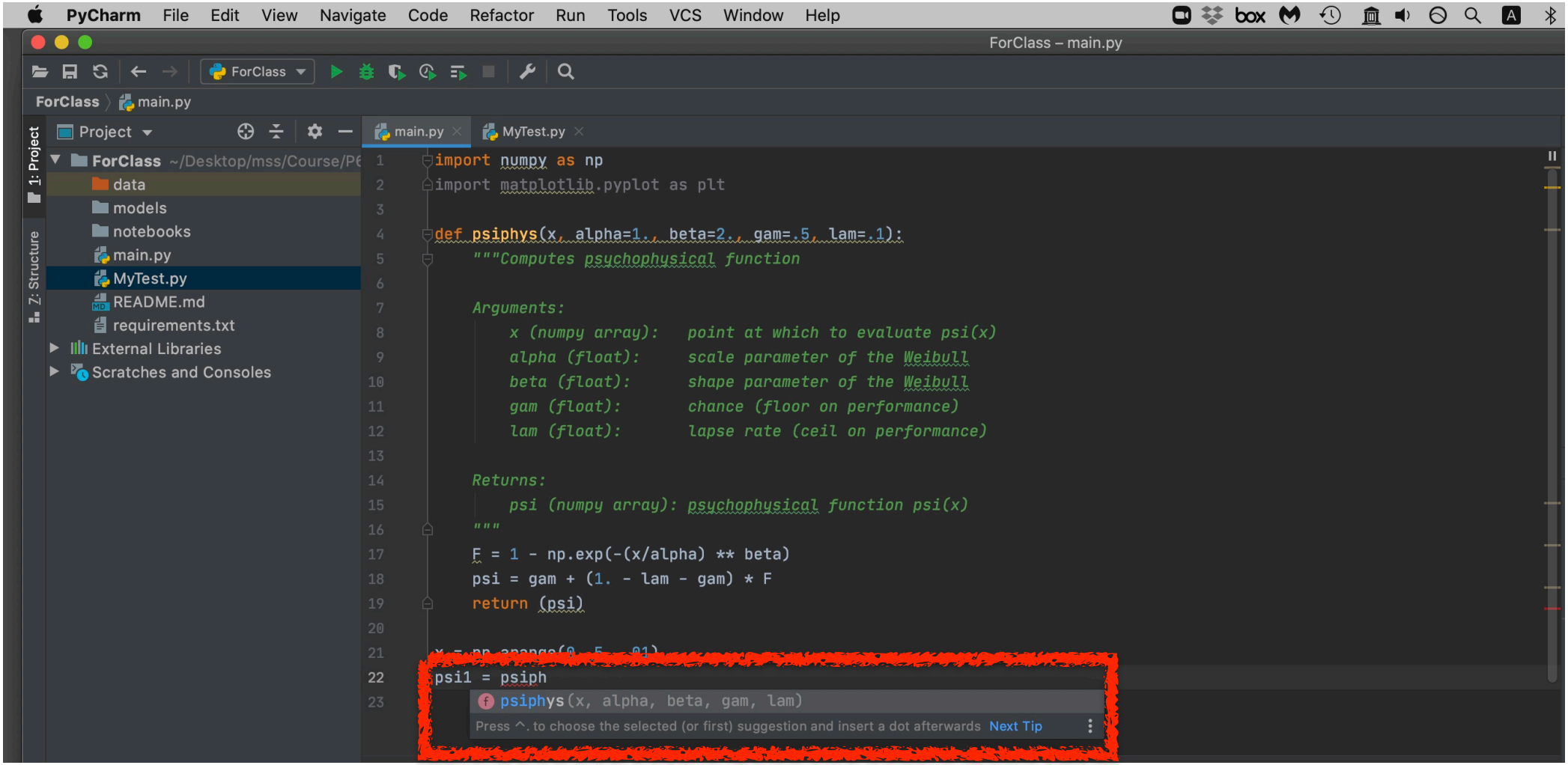
Packages installed successfully: Installed packages: 'scipy' (today 5:49 PM)



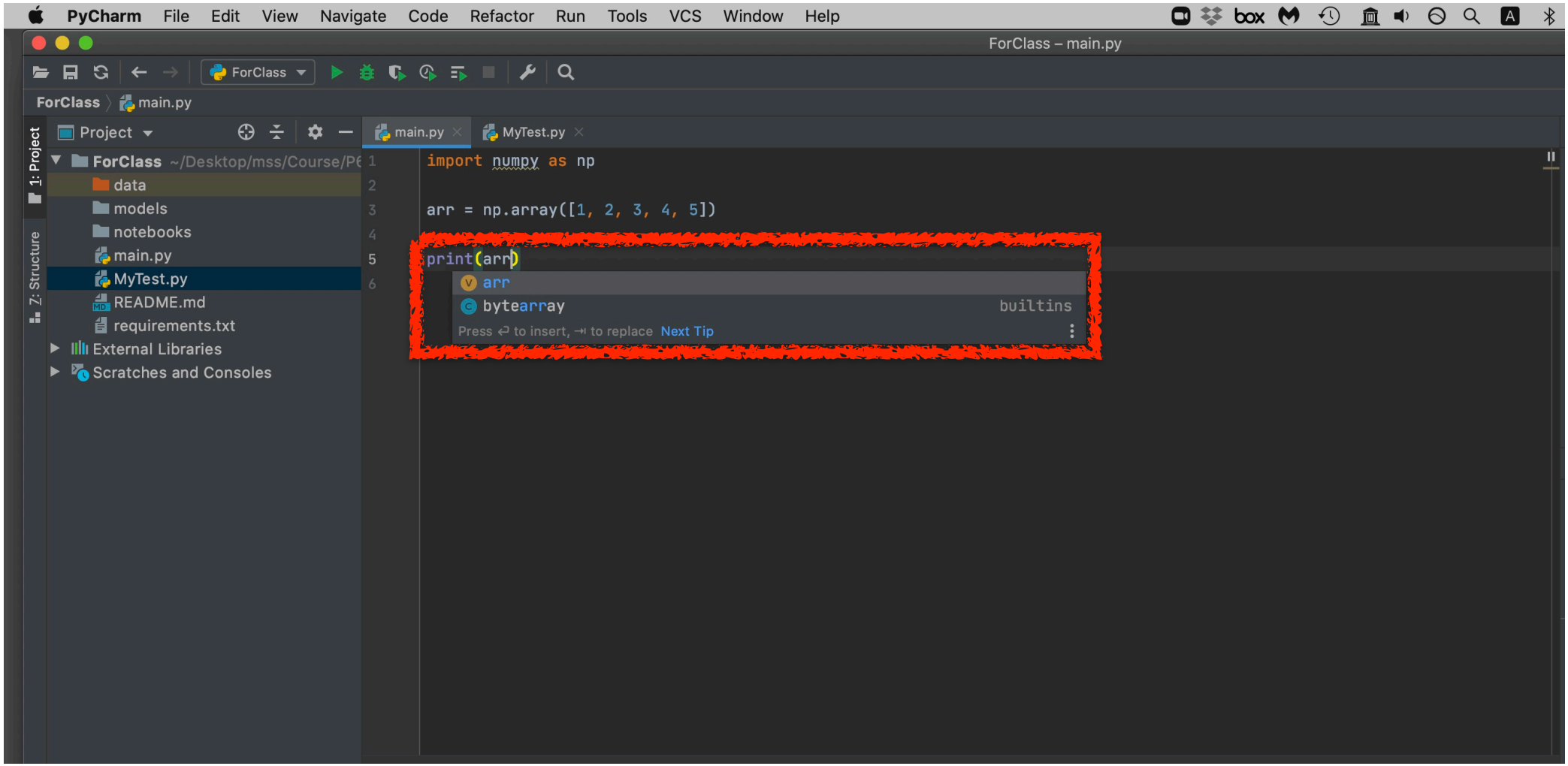
each time you run the .py file, you will create a new Console (so, you'll want to close them)

autocompletion as you're typing in code (Tab to complete, or use arrows and Enter to select)

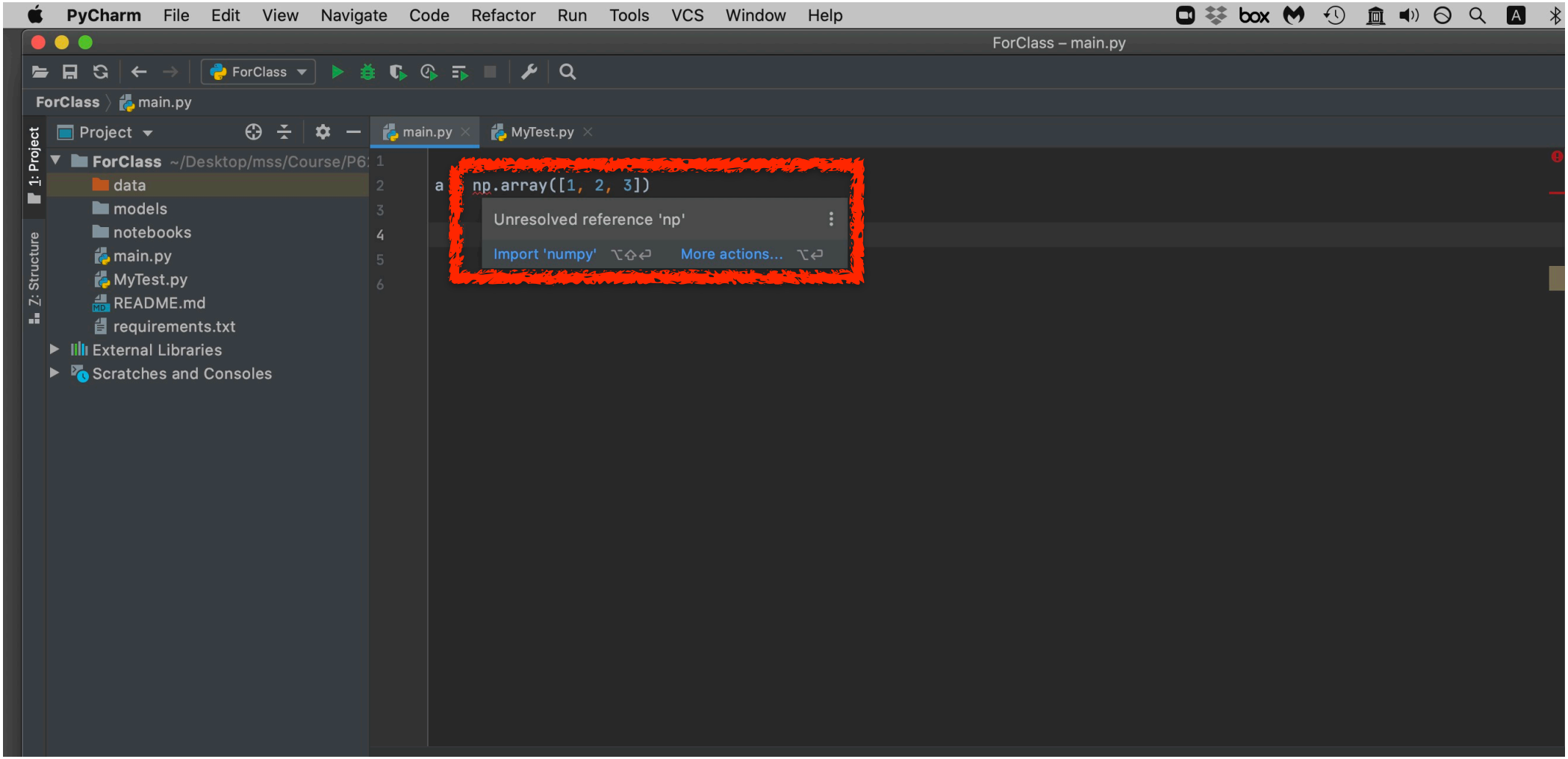




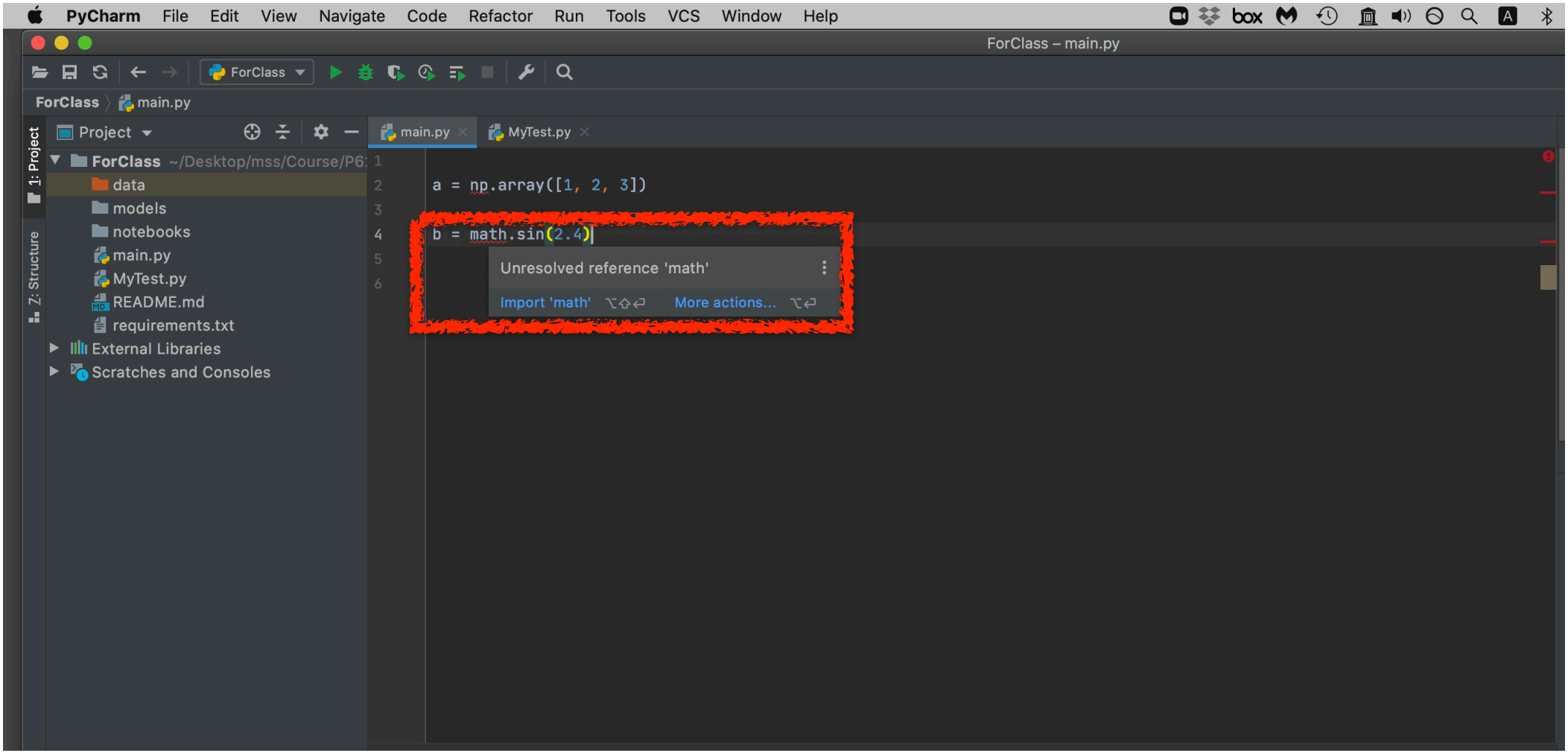
autocompletion as you're typing in code (Tab to complete, or use arrows and Enter to select)

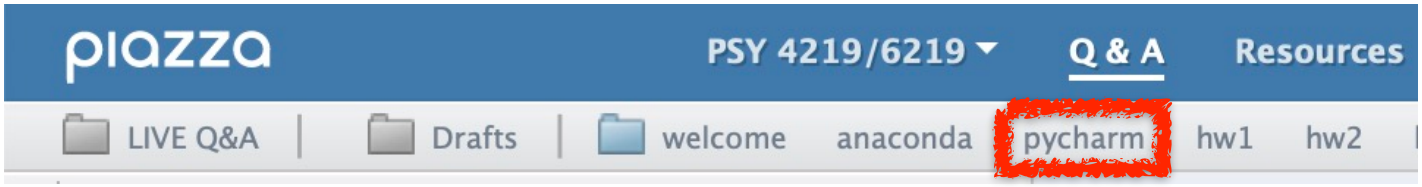

PyCharm File Edit View Navigate Code Refactor Run Tools VCS Window Help
ForClass – main.py
ForClass
ForClass 〉 main.py
Project
main.py
MyTest.py
ForClass ~/Desktop/mss/Course/P6
data
models
notebooks
main.py
MyTest.py
README.md
requirements.txt
External Libraries
Scratches and Consoles
1: Project
Z: Structure
1 import numpy as np
2
3 arr = np.array([1, 2, 3, 4, 5])
4
5 print(arr)
6
arr
bytearray builtins
Press ⏎ to insert, ⇥ to replace Next Tip

can automatically import a (missing) module (in this case numpy)



can automatically import a (missing) module (in this case math)





post questions about using PyCharm on Piazza

a module in Python

- a module in Python is most simply a .py file that contains variables and/or functions that can be called from another program using an `import` command
- follow the same rules as importing and using modules like numpy, matplotlib, only here we're talking about a module you might create and use yourself

```
import mymodule
import mymodule as mm
from mymodule import myfun, myvar
```

see `PsiPhys.py` and `TestPsiPhys.py`

test/demo within a module (`__main__`)

imports

variable definitions

function definitions

```
if __name__ == '__main__':
```

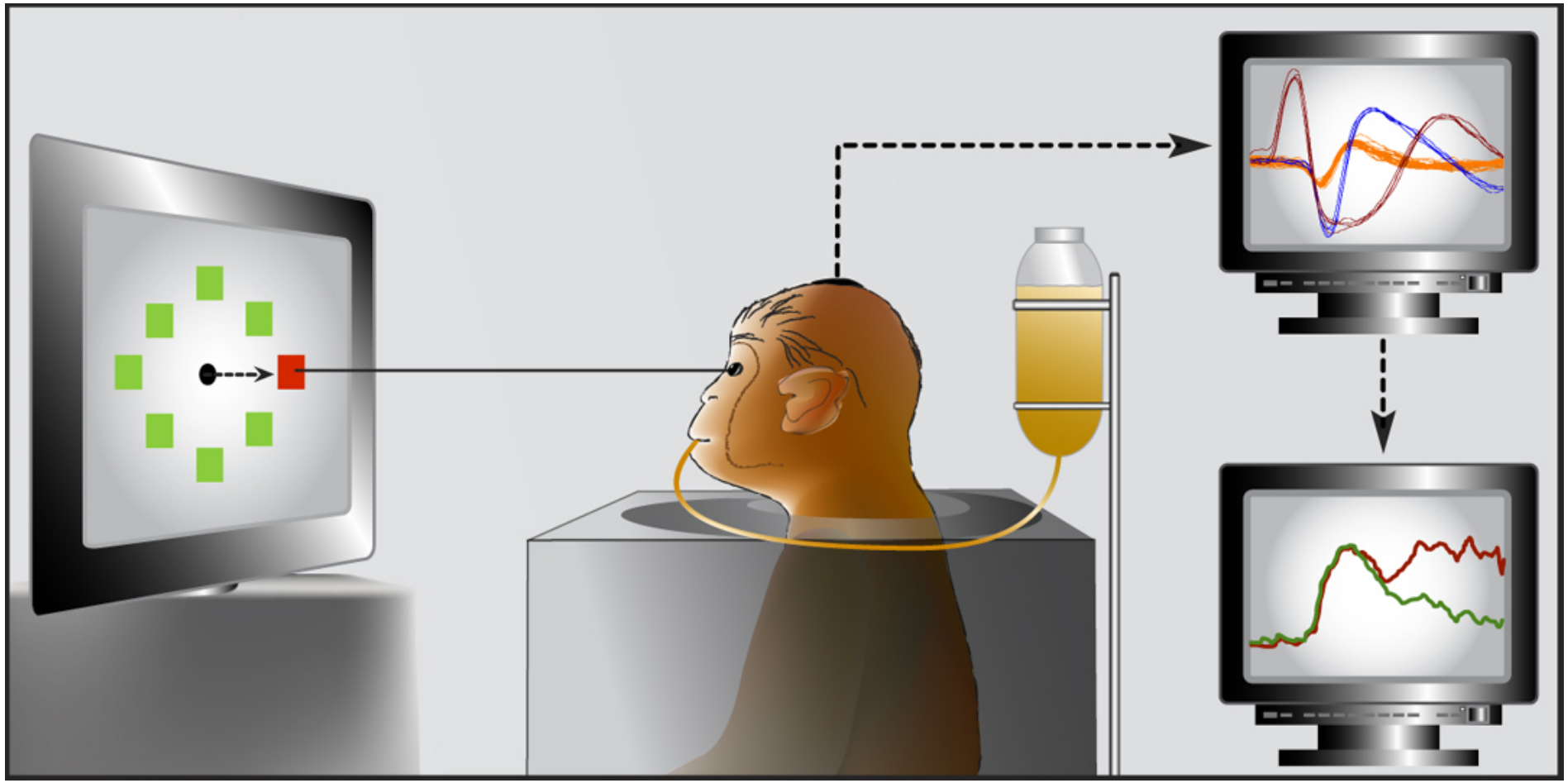
things to run to test/demo the module

this is only executed when file run on its own,
not used when imported as a module

Random Numbers

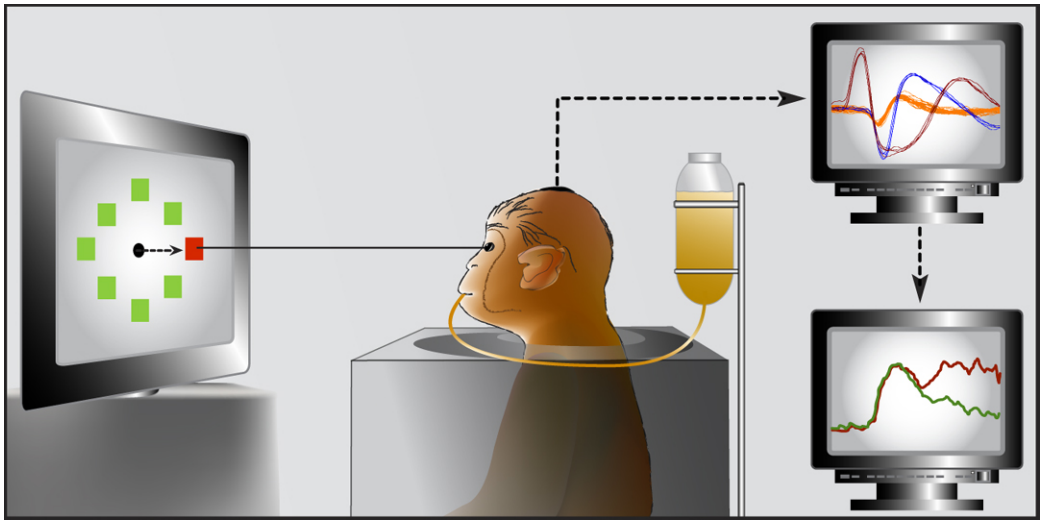
Hellekalek, P. (1998). Good random number generators are (not so) easy to find. *Mathematics and Computers in Simulation*, 46, 485-505.

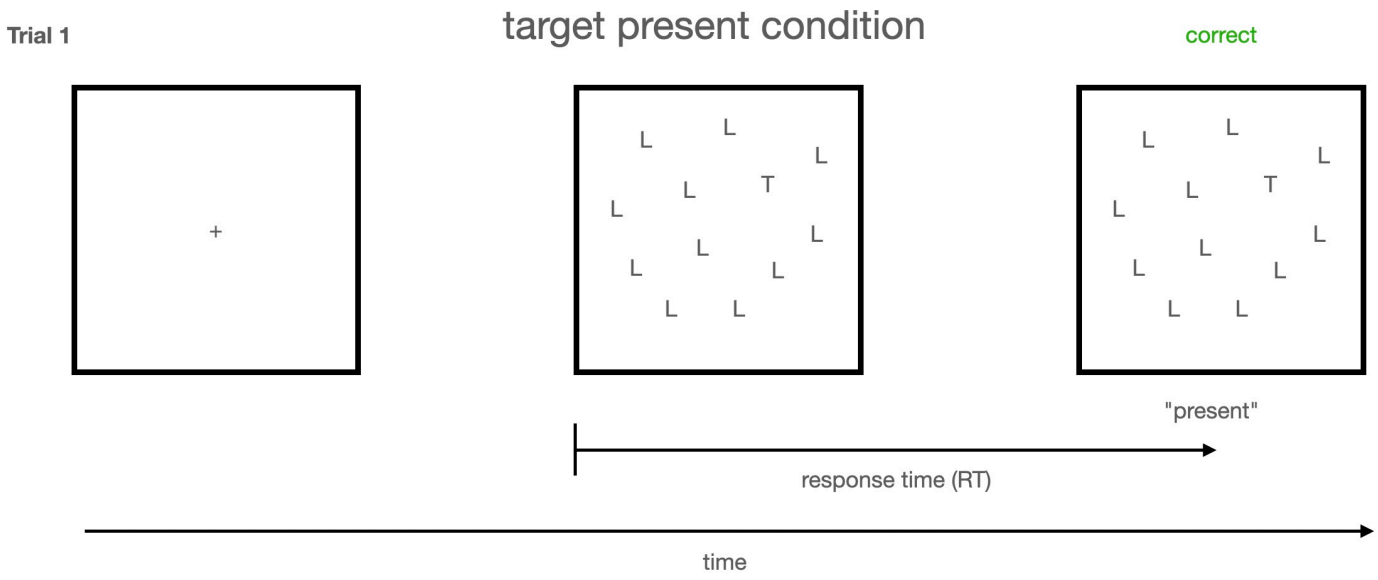
uses of random numbers (in psychology and neuroscience)



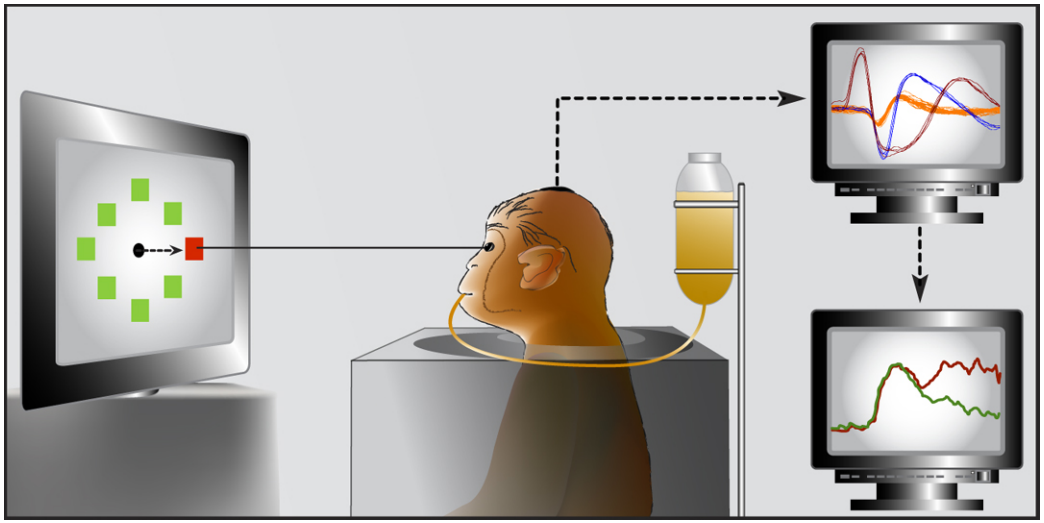
an example

randomization in a behavioral experiment



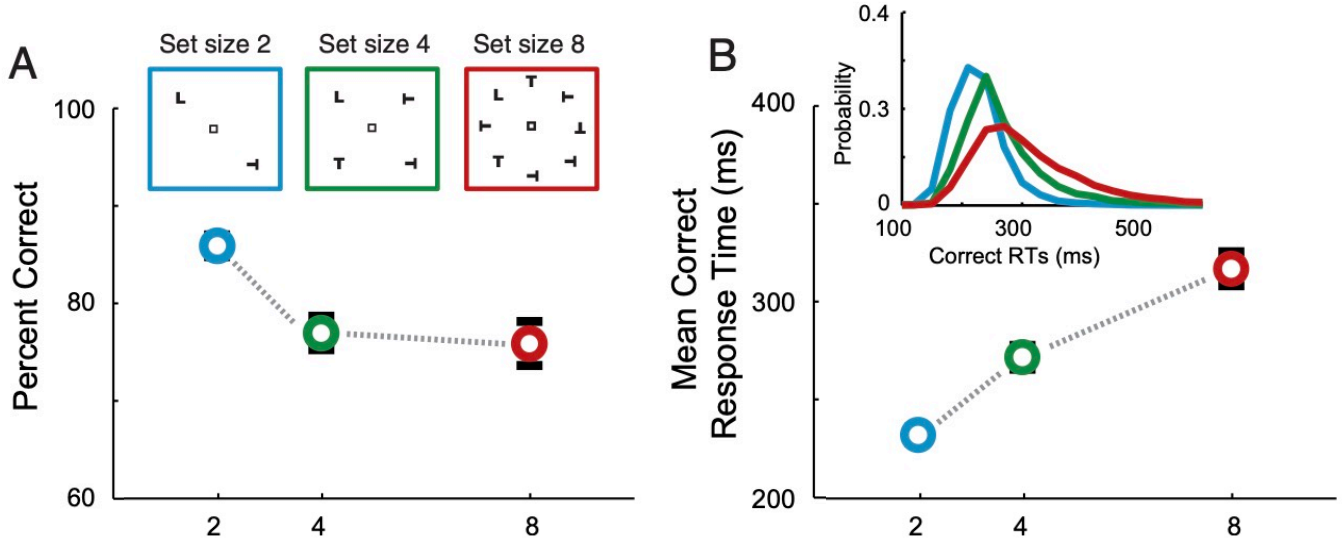


randomization in a behavioral experiment



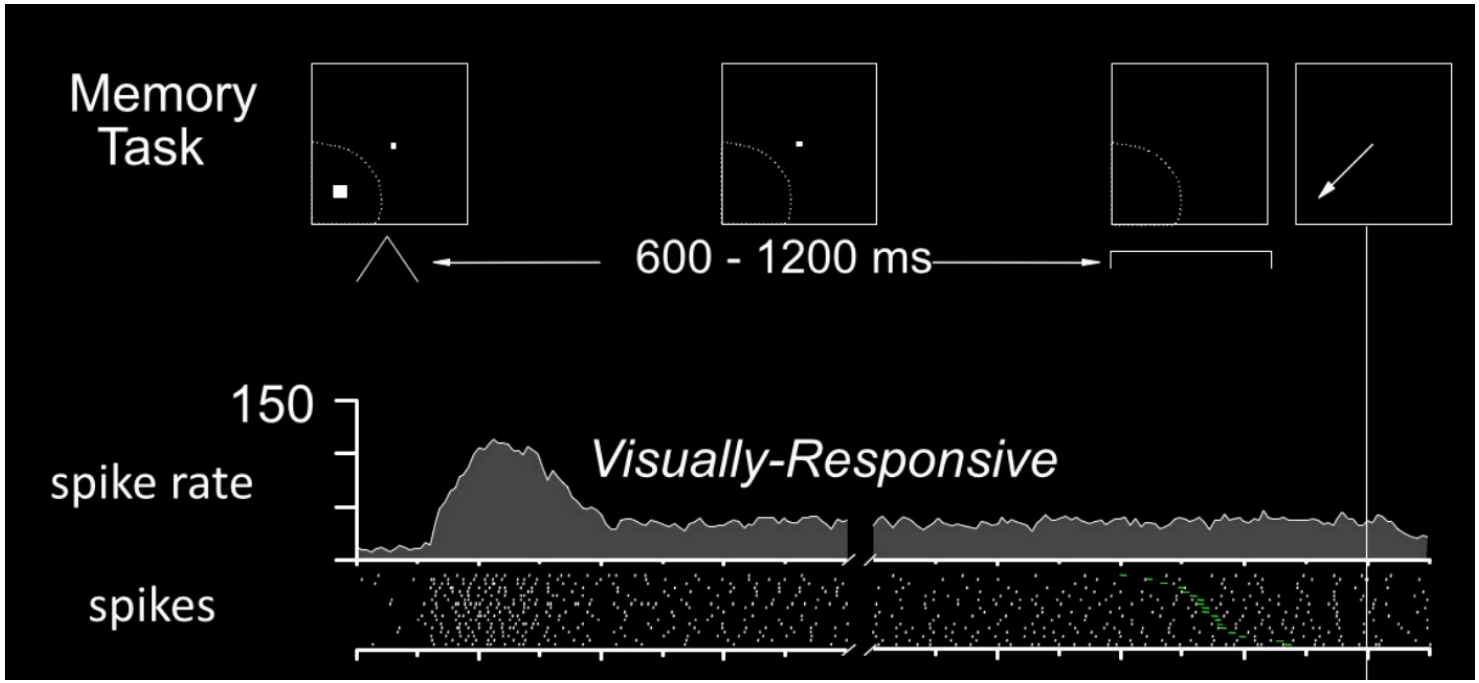
- randomize the presence/absence and the location of target and distractors in the array from trial to trial, randomize the set size from trial to trial
  - makes target location and search conditions unpredictable
- constrain to avoid repetitions of the same location from one trial to the next constrain to avoid more than three repetitions of the same set size in a row
  - breaking the pure randomization for empirical reasons
- might have a higher probability of the target appearing in particular locations over others (depending on location of neural probe)
  - breaking the pure randomization for practical reasons

randomness (variability) in behavior



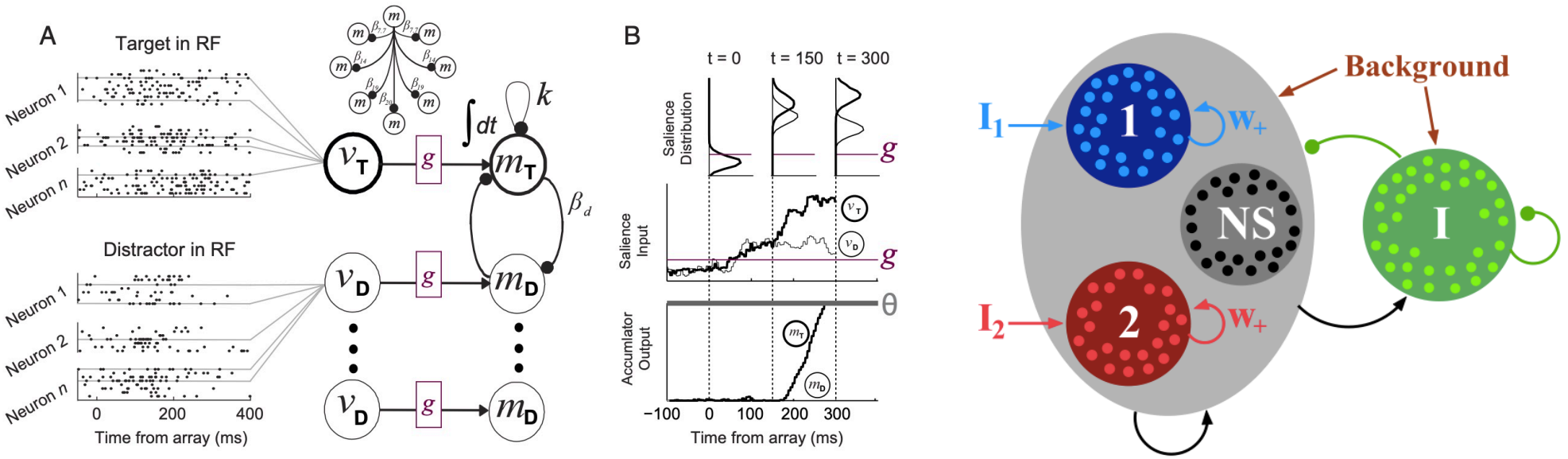
- monkeys (and humans) sometimes are correct, sometimes make errors (given the same stimulus array)
- monkeys (and humans) vary in their response times, sometimes slow, sometimes fast (given the same stimulus array)
- our models (computational and mathematical instantiations of theories) need to account for this variability (error rates and distributions of response times) with simulated randomness

randomness (variability) in the brain



- brain is "noisy", with neurons producing spikes at certain rates, that vary over time, but with tremendous variability in interspike times (simulate noise with random variables)
- randomness does not mean unpredictable or unprincipled, but means that there's variability (noise) and that variability (around some mean) cannot be predicted

randomization in a model



- to account for variability in behavior, we need models that have variability within them (given the same stimulus, a different behavior at a different time occurs)
- to account for variability in neural activity, we similarly need models that have variability within them (given the same stimulus, a different pattern of neural activity with a different time course occurs)

using randomness to fit and test models

$$P(H \mid D) = \frac{P(D \mid H)P(H)}{P(D)}$$

T. Bayes.

- we might use Bayes methods for parameter estimation (model fitting) and model comparison
- Bayesian methods make probabilistic statements about parameters and models, requiring the use of sophisticated random number generators to operate
- or we might use a genetic algorithm or simulated annealing to estimate/optimize parameters, also requiring the use of random number generators to operate

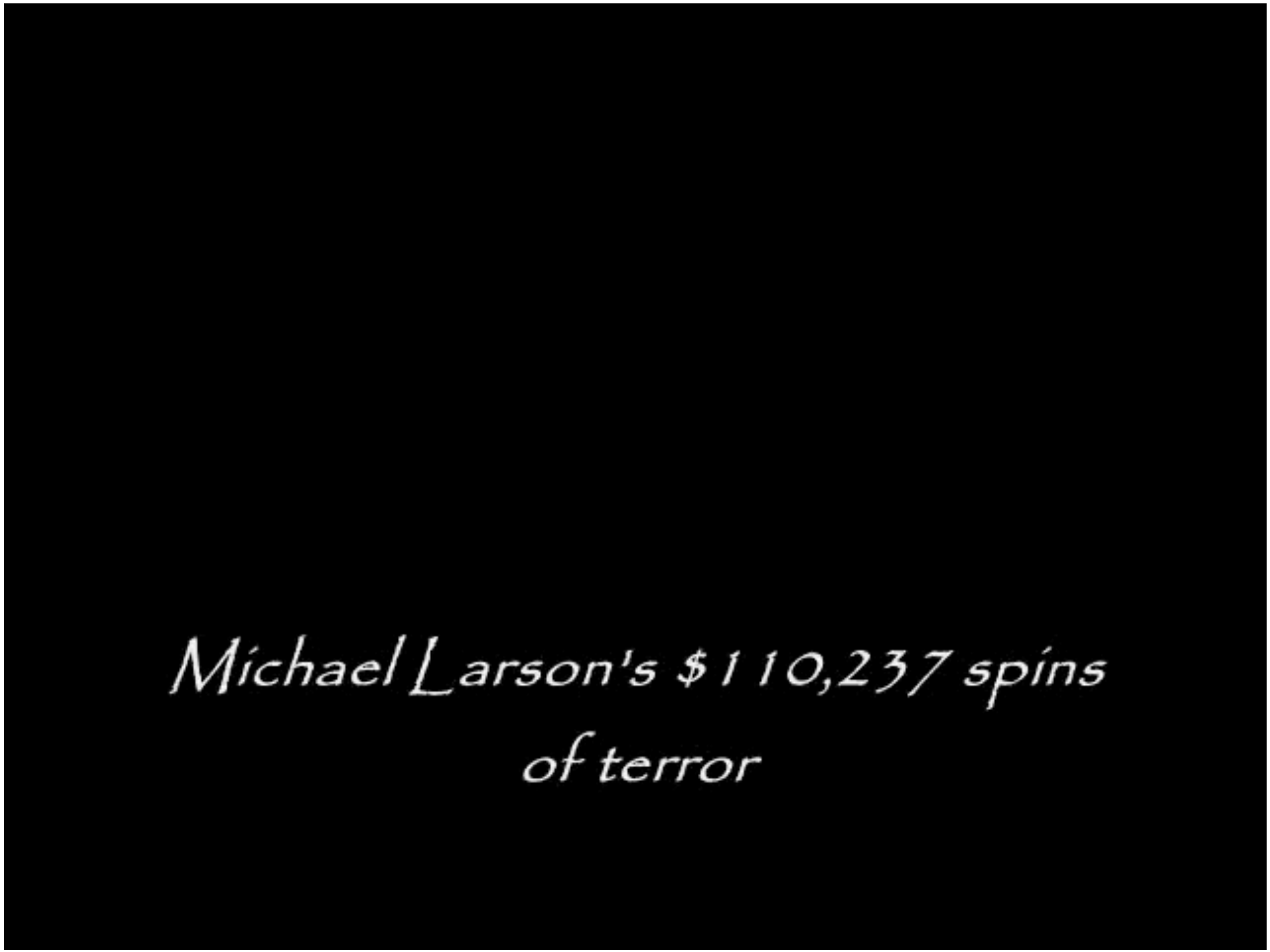
uses of random numbers

- casino games and lotteries
- randomness in video games
- statistics, distinguish real from chance differences
- simulating physical, biological, social systems
- cryptographic applications
- solving mathematical problems using random numbers

examples of failures of randomness

Press Your Luck

http://www.thisamericanlife.org/radio-archives/episode/412/million-dollar-idea
https://www.youtube.com/watch?v=bfOm7K8A0Pw


Michael Larson's $110,237 spins
of terror

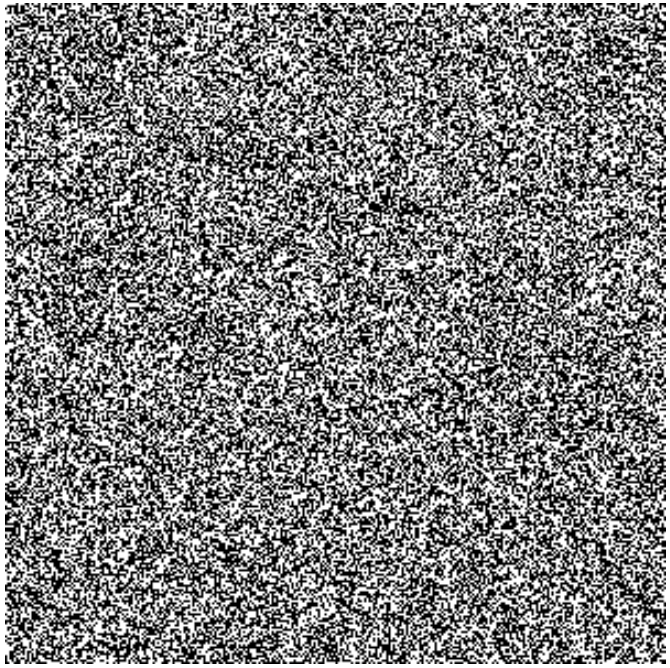
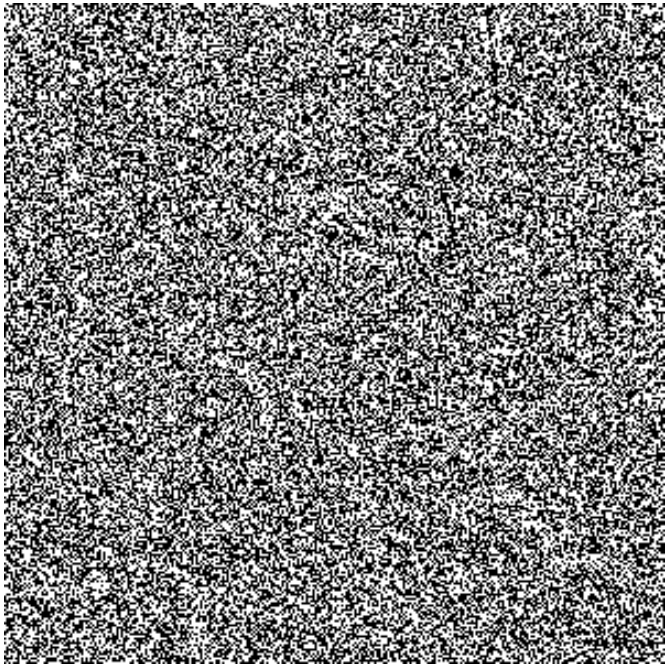
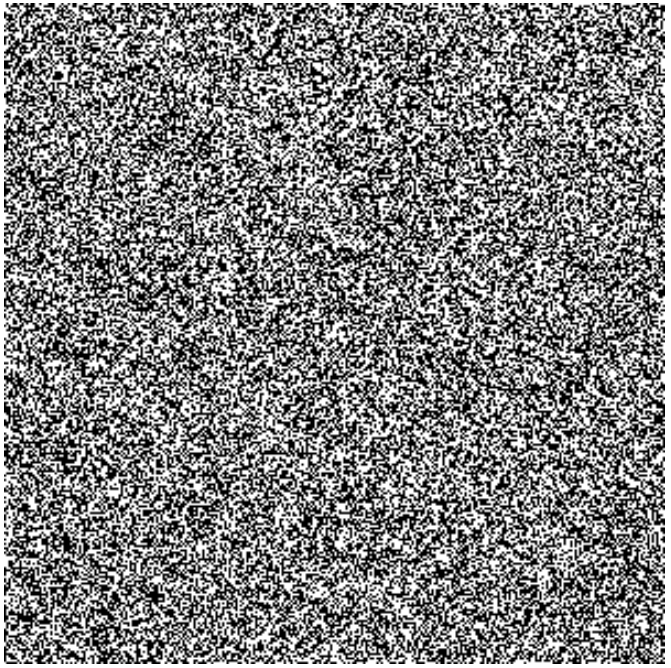
In 2007, the Tennessee lottery switched from drawing balls to a computer program for their pick-3 and pick-4 daily lottery.

After switching, no winning number ever had duplicate numbers, meaning that numbers like “667” or “5521” never came up.

It was soon discovered that this was a software error: *“a keypunch error by an employee of Smartplay International Inc. … the employee typed in a "u" for "unique" instead of an "r" for "repeat" in the computer code …”*

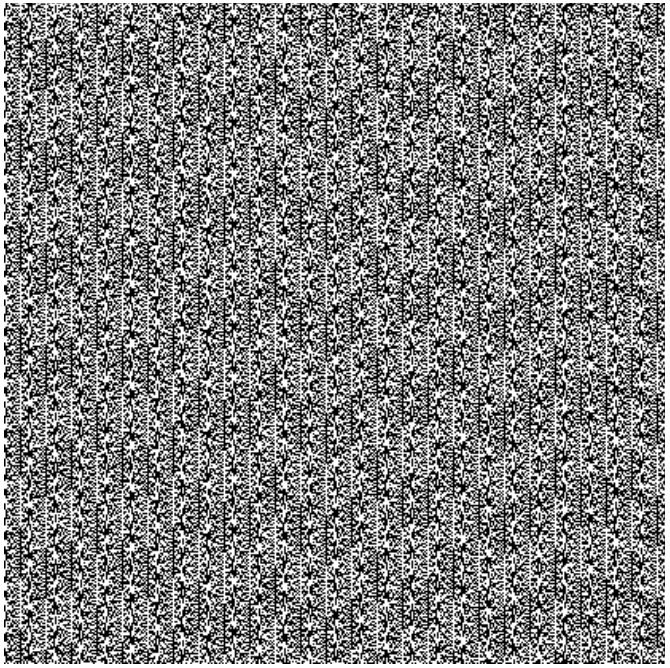
Imagine generating a random bit map image of 0’s and 1’s …

Imagine generating a random bit map image of 0’s and 1’s …

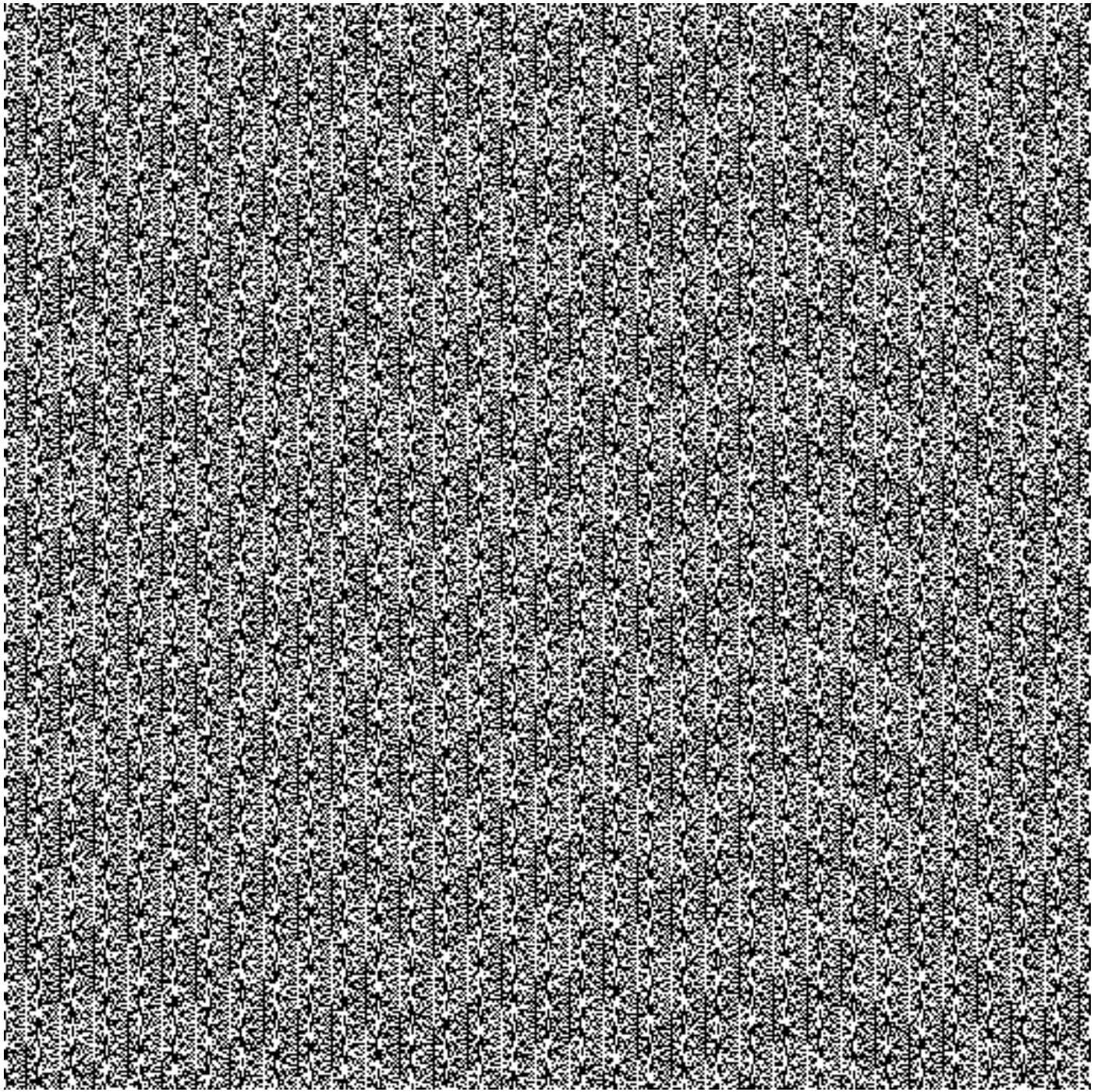
they should look something like this …

Imagine generating a random bit map image of 0’s and 1’s …

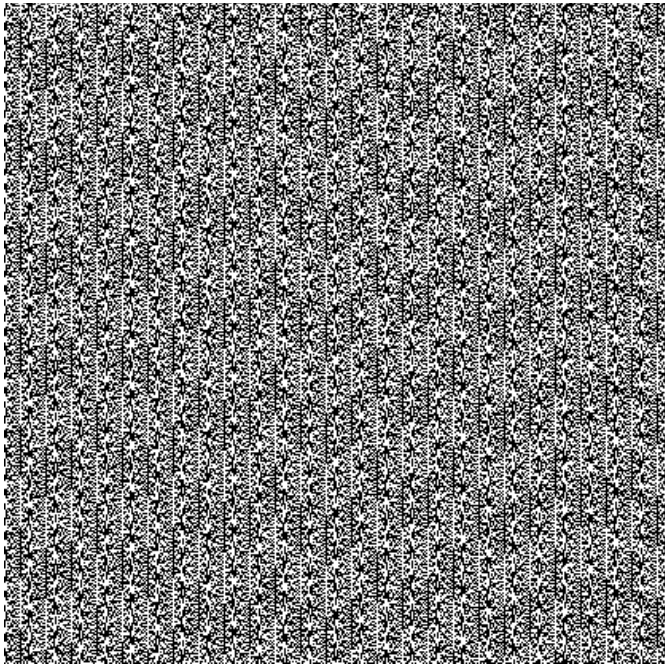


this clearly doesn’t look random …

Imagine generating a random bit map image of 0’s and 1’s …

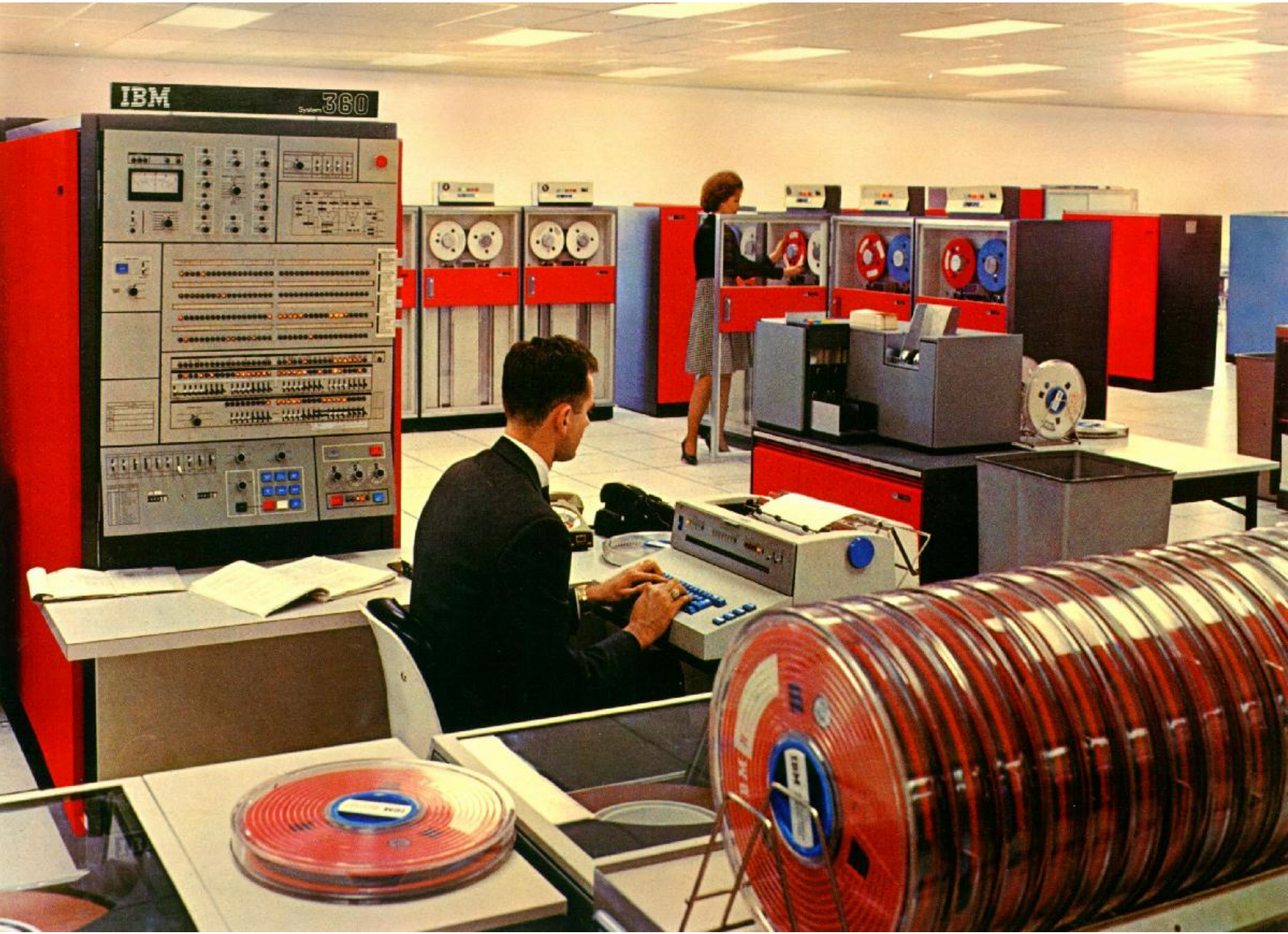


Imagine generating a random bit map image of 0's and 1's …



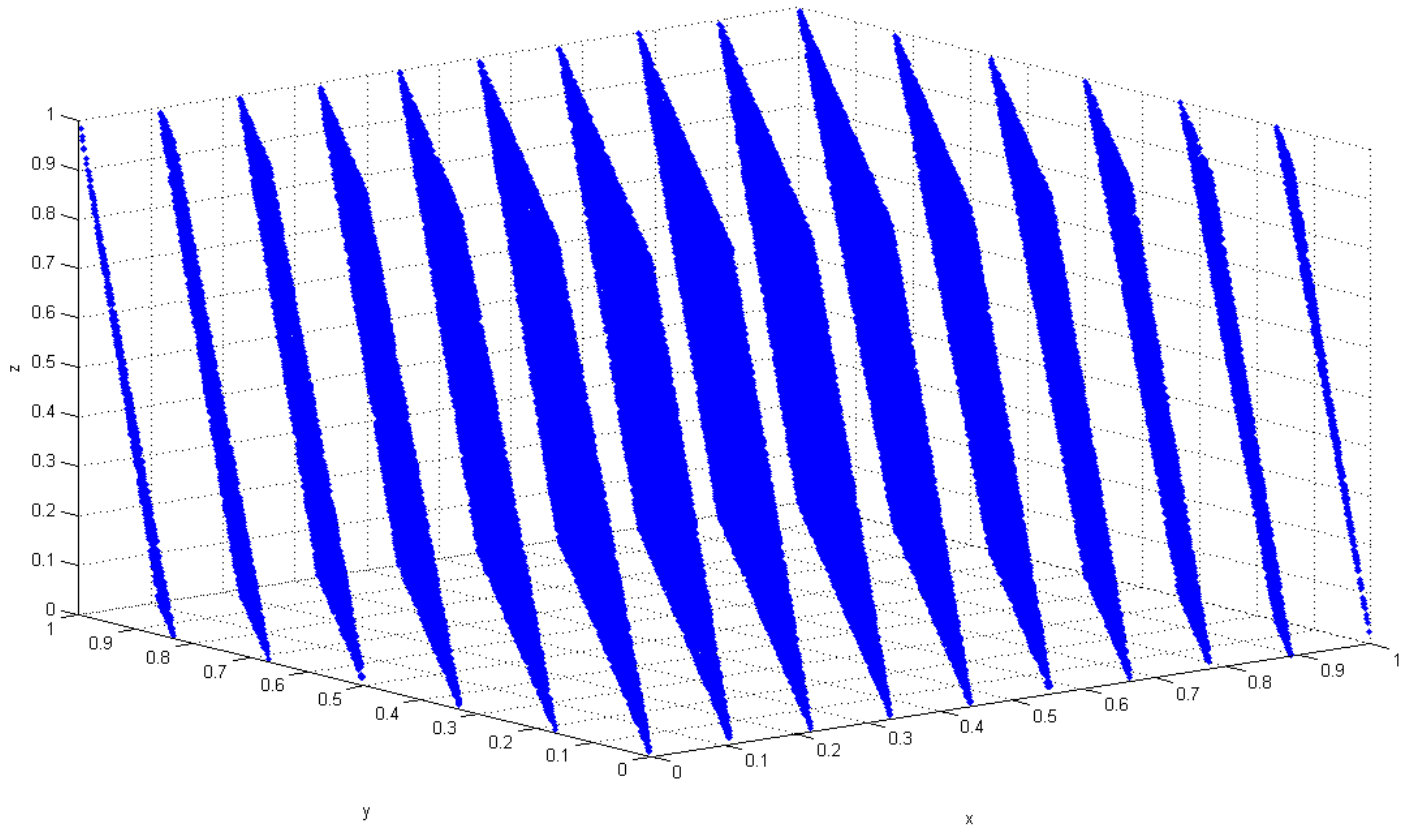
yet it was generated by the `rand()` function in PHP in Microsoft Windows in 2008 …

imagine you were using that `rand()` function for security …
and discovered that the random numbers weren't really random …


IBM
System 360

Randu - *random number generator widely used in the 1960s and 1970s on IBM 360 mainframe computers*

if the numbers were truly random, they would be in a cloud, not be well-structured on planes



plot of 1000s of sequential random samples (x,y,z)

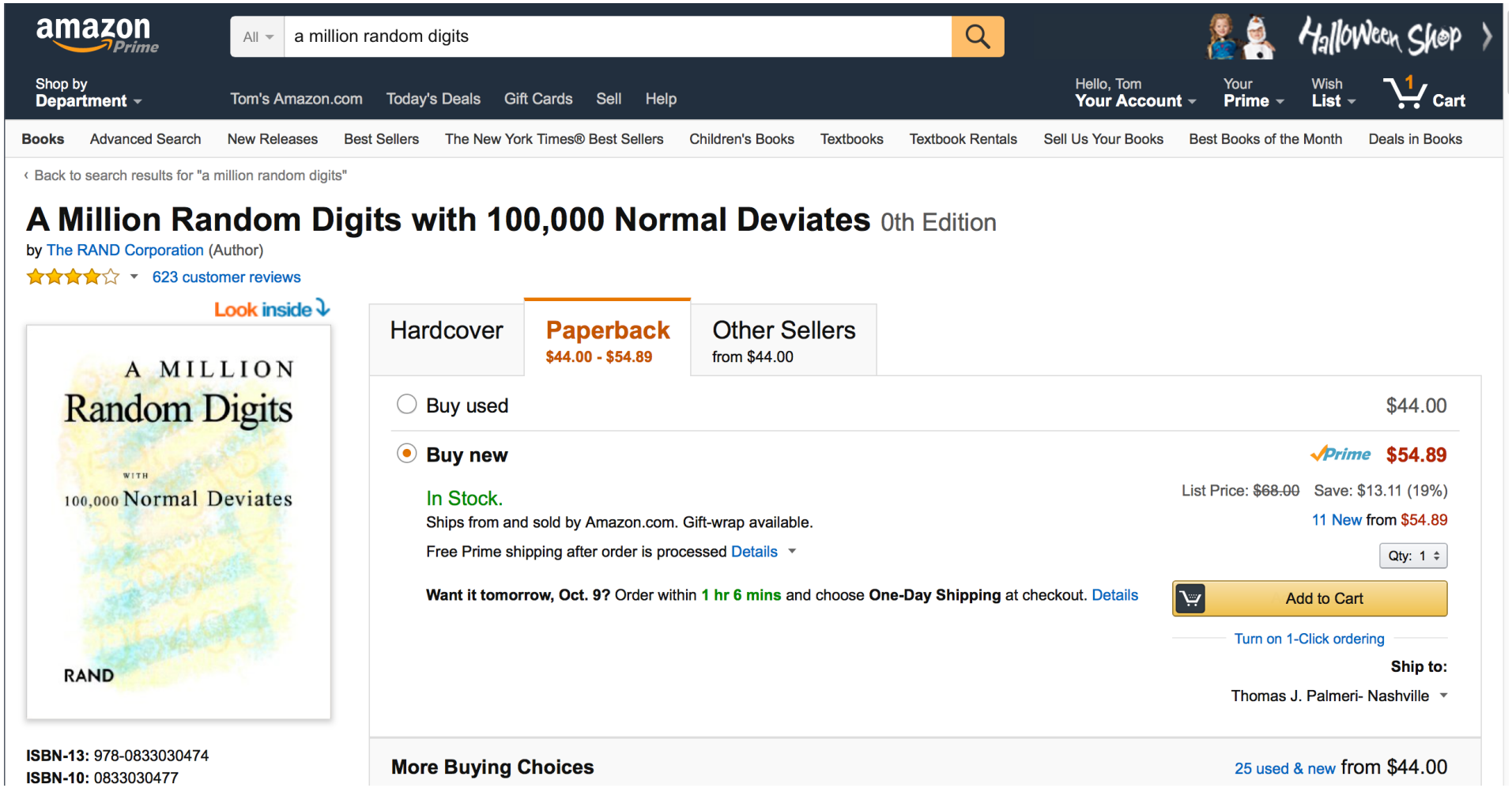
what is random?

unpredictable, not following a pattern or plan

randomness from a physical process

all balls (numbers) are (should be) equally likely

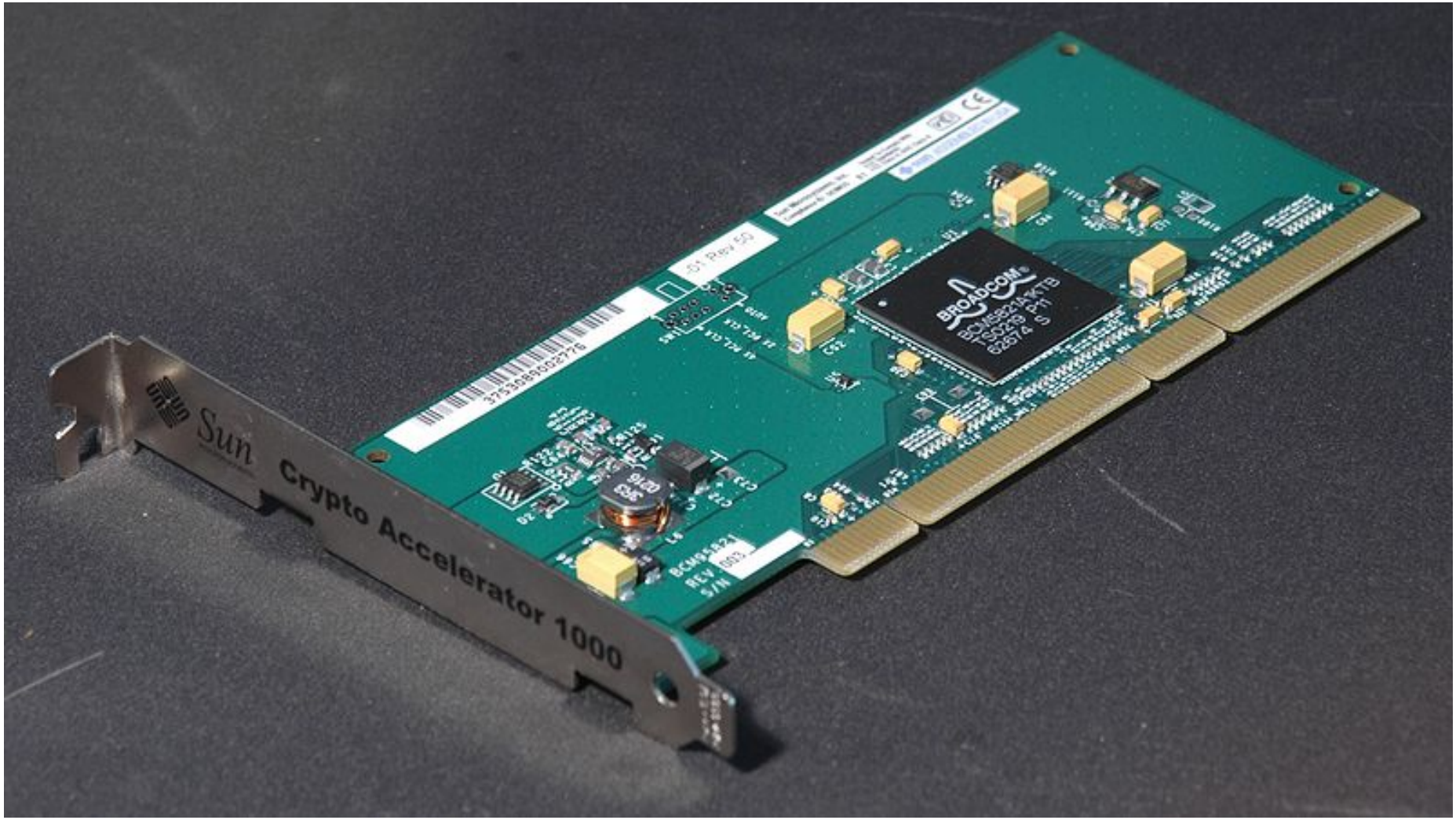
randomness from a physical process



RAND Corporation began generating random digits with an "electronic roulette wheel", consisting of a random frequency pulse source of about 100,000 pulses per second gated once per second with a constant frequency pulse and fed into a 5-bit binary counter.

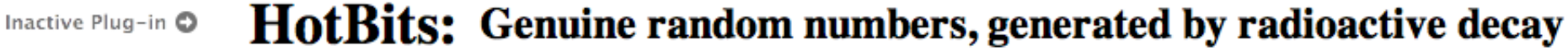
https://smile.amazon.com/Million-Random-Digits-Normal-Deviates/dp/0833030477

randomness from a physical process



randomness from a physical process

www.random.org *based on atmospheric noise levels*



www.fourmilab.ch/hotbits/ *based on radioactive decay*

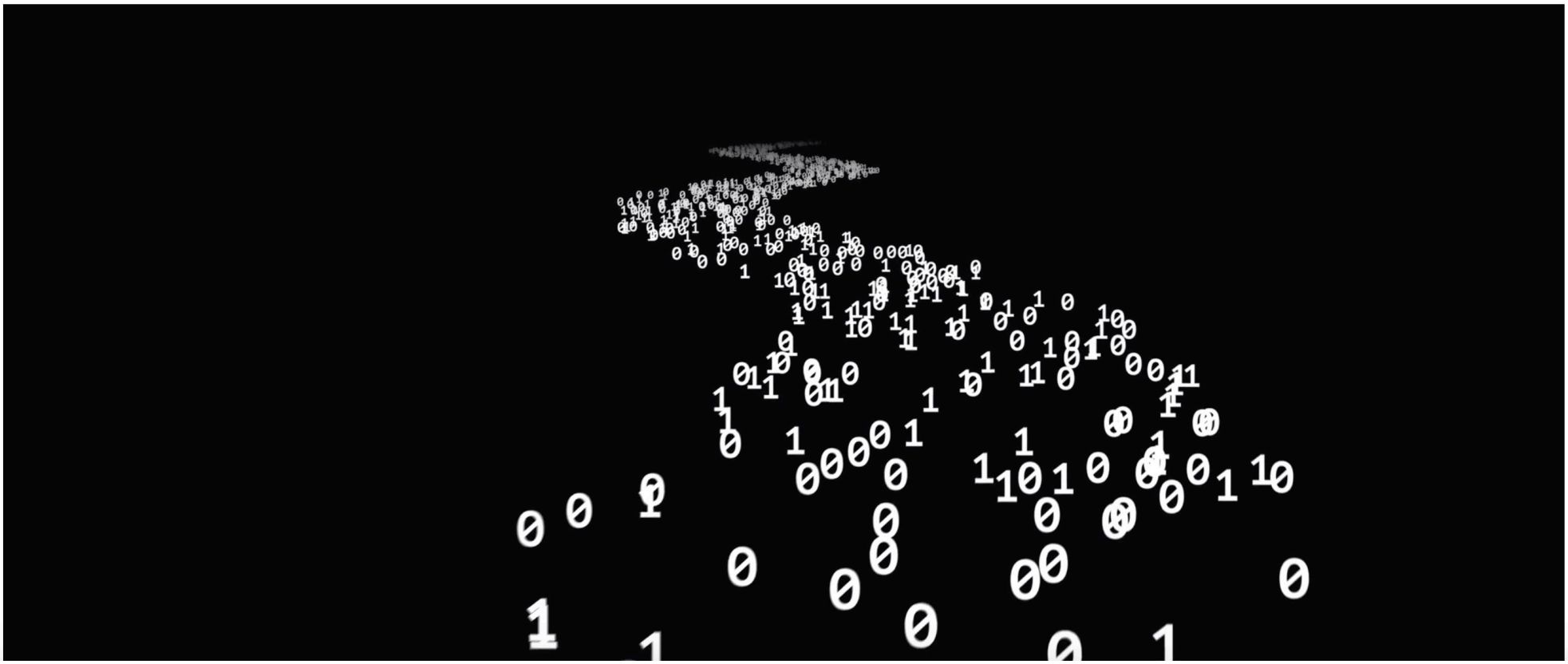
randomness from a physical process

WIRED BUSINESS CULTURE GEAR IDEAS SCIENCE SECURITY TRANSPORTATION MY ACCOUNT GIVE A GIFT

ANIL ANANTHASWAMY SCIENCE 06.23.2019 08:00 AM

Quantum Computers Could Be True Randomness Generators

Pure, verifiable randomness is essential to encryption yet hard to come by. Quantum computers could be the answer.



https://www.wired.com/story/quantum-computers-could-be-true-randomness-generators/

random numbers from a **physical process**

random number generators

random sequences of 0s and 1s: e.g., from a Bernoulli process with p=.5
like getting a heads (1) or tails (0) with fair coin

0 1 0 0 0 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 1 1 0 1 0 0 ...

random sequences of floats: e.g., from uniform distribution [0, 1)

.143 .543 .723 .334 .383 .109 .092 .913 .398 .743 ...

(without connection to a random physical process)

random numbers on a **computer**

pseudo-random number generators

random sequences of 0s and 1s: e.g., from a Bernoulli process with p=.5 like getting a heads (1) or tails (0) with fair coin

0 1 0 0 0 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 1 1 0 1 0 0 ...

random sequences of floats: e.g., from uniform distribution [0, 1)

.143 .543 .723 .334 .383 .109 .092 .913 .398 .743 ...

(without connection to a random physical process)

random numbers on a computer

pseudo-random number generators

future quantum computers will have a physical randomization process

- every computerized random number generator is the complete opposite from random : a completely deterministic algorithm

- every pseudo-random number generator is cyclical; eventually the deterministic sequence repeats itself

- a good pseudo-random number generator produces sequences of numbers that have the same statistical properties as sequences from a truly random processes

- determinism can be a good thing when you want to exactly recreate some simulation or analysis that uses random numbers

(without connection to a random physical process)

random numbers on a computer

pseudo-random number generators

- every computerized random number generator is the complete opposite from random : a completely deterministic algorithm

0100010100…101011111110101…0101011110100…00110011111 …

if I know I'm here in the sequence,
the rest of the sequence is perfectly predictable

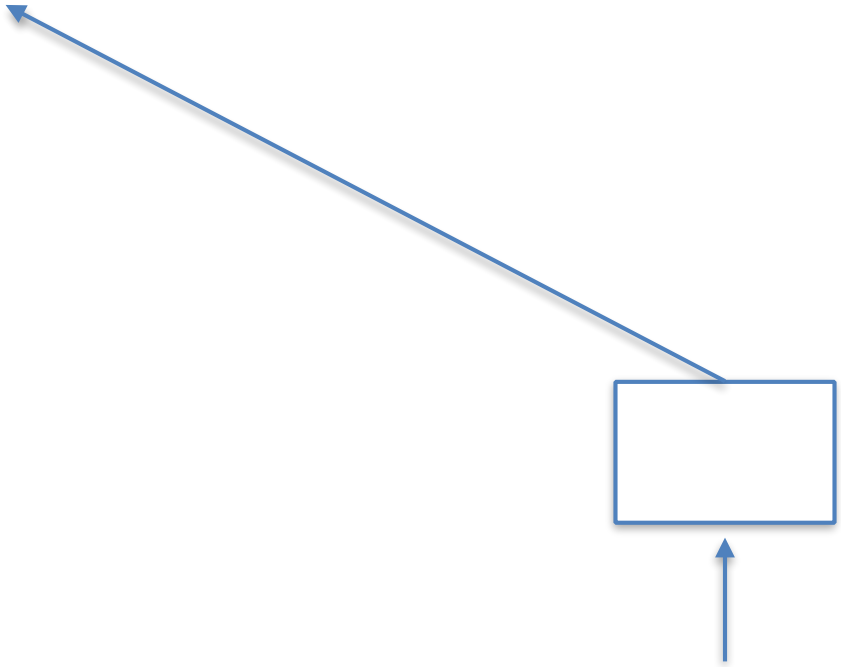
(without connection to a random physical process)

random numbers on a computer

pseudo-random number generators

- every computerized random number generator is the complete opposite from random : a completely deterministic algorithm

0100010100…101011111110101…0101011110100…00110011111 …



seed = 42421234

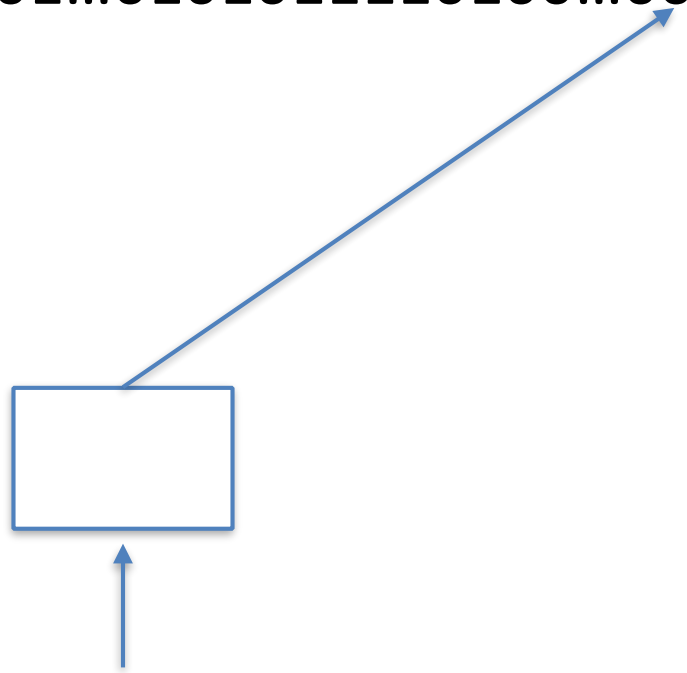
(without connection to a random physical process)

random numbers on a computer

pseudo-random number generators

- every computerized random number generator is the complete opposite from random : a completely deterministic algorithm

0100010100…101011111110101…0101011110100…00110011111 …



seed = 982321232

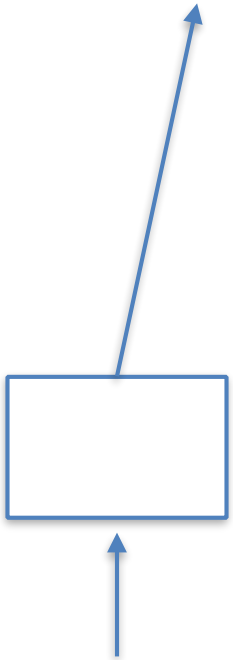
(without connection to a random physical process)

random numbers on a computer

pseudo-random number generators

- every computerized random number generator is the complete opposite from random : a completely deterministic algorithm

0100010100…101011111110101…0101011110100…00110011111 …



seed = 213445213

(without connection to a random physical process)

random numbers on a computer

pseudo-random number generators

- every pseudo-random number generator is cyclical; eventually the deterministic sequence repeats itself

0100010100…101011111110101…0101011110100…00110011111 …

a very poor PRNG might repeat after 2^{16}-1 random numbers;
a simulation needing millions of random numbers would actually be using a cyclical function of numbers instead

(without connection to a random physical process)

random numbers on a computer

pseudo-random number generators

- nearly every pseudo-random number generator is cyclical; eventually the deterministic sequence repeats itself

0100010100…101011111110101…0101011110100…00110011111 …

a good one, like the ones used in Python, might take longer than the current age of the universe to see the cycle repeat when printing out trillions of numbers every second

(without connection to a random physical process)

random numbers on a computer

pseudo-random number generators

- a good pseudo-random number generator produces sequences of numbers that have the same statistical properties as sequences from a truly random processes

0100010100…101011111110101…0101011110100…00110011111 …

pseudo-random number generator OR real random process

(without connection to a random physical process)

random numbers on a computer

pseudo-random number generators

- a good pseudo-random number generator produces sequences of numbers that have the same statistical properties as sequences from a truly random processes

0100010100…101011111110101…0101011110100…00110011111 …

pseudo-random number generator OR real random process

1001111001…011111110100011…1111000011101…11000110010 …

pseudo-random number generator OR real random process

(without connection to a random physical process)

random numbers on a computer

pseudo-random number generators

- a good pseudo-random number generator produces sequences of numbers that have the same statistical properties as sequences from a truly random processes

0100010100…10101111110101…0101011110100…00110011111 …

pseudo-random number generator OR real random process

1001111001…011111110100011…1111000011101…11000110010 …

pseudo-random number generator OR real random process

1011100111…110000010100111…0001111011111…01010000101 …

pseudo-random number generator OR real random process

no better than a coin flip to decide (if a good pseudo-random number generator)

a few (of many) tests for randomness

NIST National Institute of Standards and Technology
Information Technology Laboratory

SEARCH CSRC: GO

ABOUT MISSION CONTACT STAFF SITE MAP

Computer Security Division
Computer Security Resource Center

CSRC HOME GROUPS PUBLICATIONS DRIVERS NEWS & EVENTS ARCHIVE

- Random Number Generation
- Download Documentation and Software
- Activities
- Publications and Presentations
- Guide to Statistical Tests
- Frequency (Monobits) Test
- Test for Frequency within a Block
- Random Binary Matrix Rank Test
- Test for the Longest Run of Ones in a Block
- Discrete Fourier Transform (Spectral) Test
- Non-overlapping (Aperiodic) Template Matching Test
- Overlapping (Periodic) Template Matching Test
- Maurer's Universal Statistical Test
- Linear Complexity Test
- Serial Test
- Approximate Entropy Test
- Cumulative Sum (Cusum) Test
- Random Excursions Test
- Random Excursions Variant Test
- References
- RNG Standards Development Bodies
- RNG and Testing Technical Working Group
- Batteries of Statistical Tests for RNG

CSRC HOME > GROUPS > ST > CRYPTOGRAPHIC TOOLKIT

GUIDE TO THE STATISTICAL TESTS

A total of fifteen statistical tests were developed, implemented and evaluated. The following describes each of the tests.

Back to Top

Frequency (Monobits) Test

Description: The focus of the test is the proportion of zeroes and ones for the entire sequence. The purpose of this test is to determine whether that number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence. The test assesses the closeness of the fraction of ones to ½, that is, the number of ones and zeroes in a sequence should be about the same.

Back to Top

Test For Frequency Within A Block

Description: The focus of the test is the proportion of zeroes and ones within M-bit blocks. The purpose of this test is to determine whether the frequency of ones is an M-bit block is approximately M/2.

Back to Top

Runs Test

Description: The focus of this test is the total number of zero and one runs in the entire sequence, where a run is an uninterrupted sequence of identical bits. A run of length k means that a run consists of exactly k identical bits and is bounded before and after with a bit of the opposite value. The purpose of the runs test is to determine whether the number of runs of ones and zeros of various lengths is as expected for a random sequence. In particular, this test determines whether the oscillation between such substrings is too fast or too slow.

Back to Top

Test For The Longest Run Of Ones In A Block

Description: The focus of the test is the longest run of ones within M-bit blocks. The purpose of this test is to determine whether the length of the longest run of ones within the tested sequence is consistent with the length of the longest run of ones that would be expected in a random sequence. Note that an irregularity in the expected length of the longest run of ones implies that

http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf

random numbers are critical for national security, financial, scientific, engineering applications

a few (of many) tests for randomness

constitute a series of statistical tests … do the random number generators produce sequences with the properties expected of the distribution they are simulating (within some confidence interval)

Monobits test : if you generate a long sequence of random 0's and 1's, do you end up with a (statistically) equal number of 0's and 1's in the long run?

Runs test : if any random sequence of 0's and 1's, you expect to find runs, as in 00011011111010011000000101100011111101
are the length and frequency of runs as expected by chance?

Fourier spectral test : Use the Fourier transform to look for periodic features in the random numbers that should not be there is the numbers are truly random

all of these entail statistical tests …

different ways of generating random numbers in Python

1) using the `random` module in base Python - NOT RECOMMENDED

2) `random` module in numpy, using **seed/state-based approach**
   - what has been used for many years
   - what you find across the web
   - what most existing Python code uses
   - not "pythonic" in style
   - does not work with parallel code
   - frozen, considered legacy
   - but it is what I will start with

3) `random` module in numpy, using **generator object approach**
   - what we will talk about later
   - recommended for code going forward

`numpy`, `scipy` use generator objects, `scikit-learn` uses seed/state approach, `keras/tensorflow` use their own random number generators

see `Random.ipynb`

random numbers in Python

one random number generator (building block) in Python is:

Mersenne twister

- developed in 1997
- period of $2^{19937}-1$
- has good statistical properties
- reasonably fast
- used in Python, Matlab
- http://en.wikipedia.org/wiki/Mersenne_twister

but even this is not appropriate for cryptographic applications
despite its statistical properties, it's possible to "crack" it

random numbers in Python (numpy)

```
import numpy.random as R

# using Matlab style prng

# sample from uniform distribution
a = R.rand()
b = R.rand(3,4)

# sample from normal (mean 0, stdev 1)
c = R.randn()
d = R.randn(6,2)
```

random numbers in Python (numpy)

```
import numpy.random as R

# using Python/numpy style prng

# sample from uniform distribution
a = R.uniform()
b = R.uniform(size=(3,4))

# sample from normal (mean 0, stdev 1)
c = R.normal()
d = R.normal(size=(6,2))
```

tuple

tuple

seeding

needs to be an integer

```
R.seed(1243134)
print(R.uniform(), R.uniform(), R.uniform())

R.seed(1243134)
print(R.uniform(), R.uniform(), R.uniform())
```

produce the same random numbers as above

```
R.seed(8237234)
print(R.uniform(), R.uniform(), R.uniform())
```

produce different random numbers

Python vs. Matlab (at start)

Python

```
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.5793889964866393
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.28131924752039117
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.4491634596890828
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.80913083038138 97
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.9150216172826635
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.4101784869762677
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.5160396404747076
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.290813502963848
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.3187345040516294
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.3072158336233559
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.12939408577526568
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.37874319118786026
(base) Tom-MacBook-Pro-2020:ForClass palmerit$ python rnd.py
0.8573560924779818
(base) Tom-MacBook-Pro-2020:ForClass palmerit$
```

Python starts with a different seed each time it is run (seeds with system time)

Python vs. Matlab (at start)

Matlab

```
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$ /Applications/MATLAB_R2019b.app/bin/matlab -nodisplay -nosplash -nojvm -batch "printrand1()"
Random Number: 0.814724
Tom-MacBook-Pro-2019:matlab palmerit$
```

Matlab starts with the same seed each time it is run

seeding

```
seed = 2754332
R.seed(seed)
a = R.uniform(size=(5, 4))
```

Best Practices

- for every application in psychology and neuroscience (and scientific computing in general) you should **ALWAYS** seed and **SAVE** the seed (reproduce your results)
- only seed **ONCE**; seeding multiple times can actually disrupt the statistical properties of a PRNG

what could the seed be?

- some function of the subject and session number (ensure that every subject/session has a unique random order)

```
seed = 9843*subject + 43*session + 93
R.seed(seed)
```

- a fixed number (if doing a long simulation)

```
seed = 5433234
R.seed(seed)
```

- date/time but save the date/time (in a file) and label as seed

```
import time
seed = time.time_ns()%(2**32 - 1)
R.seed(seed)
```

returns nanoseconds since January 1970

returns remainder

this is the max seed value

what does it mean to generate a random number

- it makes no sense to say you are "generating a random number" without specifying what probability distribution that random number is randomly sampled from

  numpy `R.uniform()` draws a random number from a continuous uniform distribution on `[0,1)`

  all (representable) real numbers on the interval `[0,1)` are equally likely