

# 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 weeks, on Wed Oct 26  
(of course, you do not need to work on it over fall break)

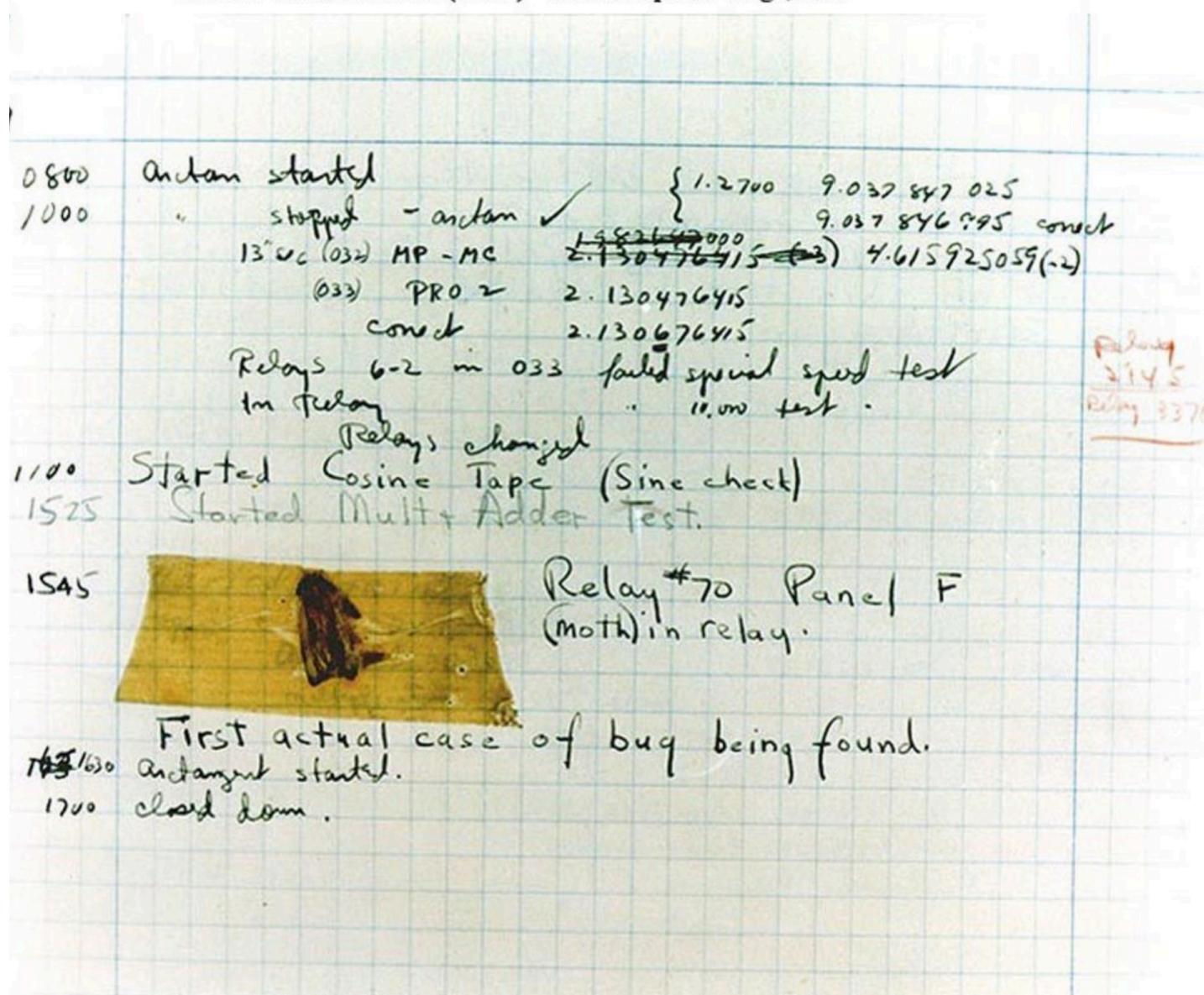
*it will be a little bit longer since you have a  
few more regular class days to work on it*

download from Brightspace

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

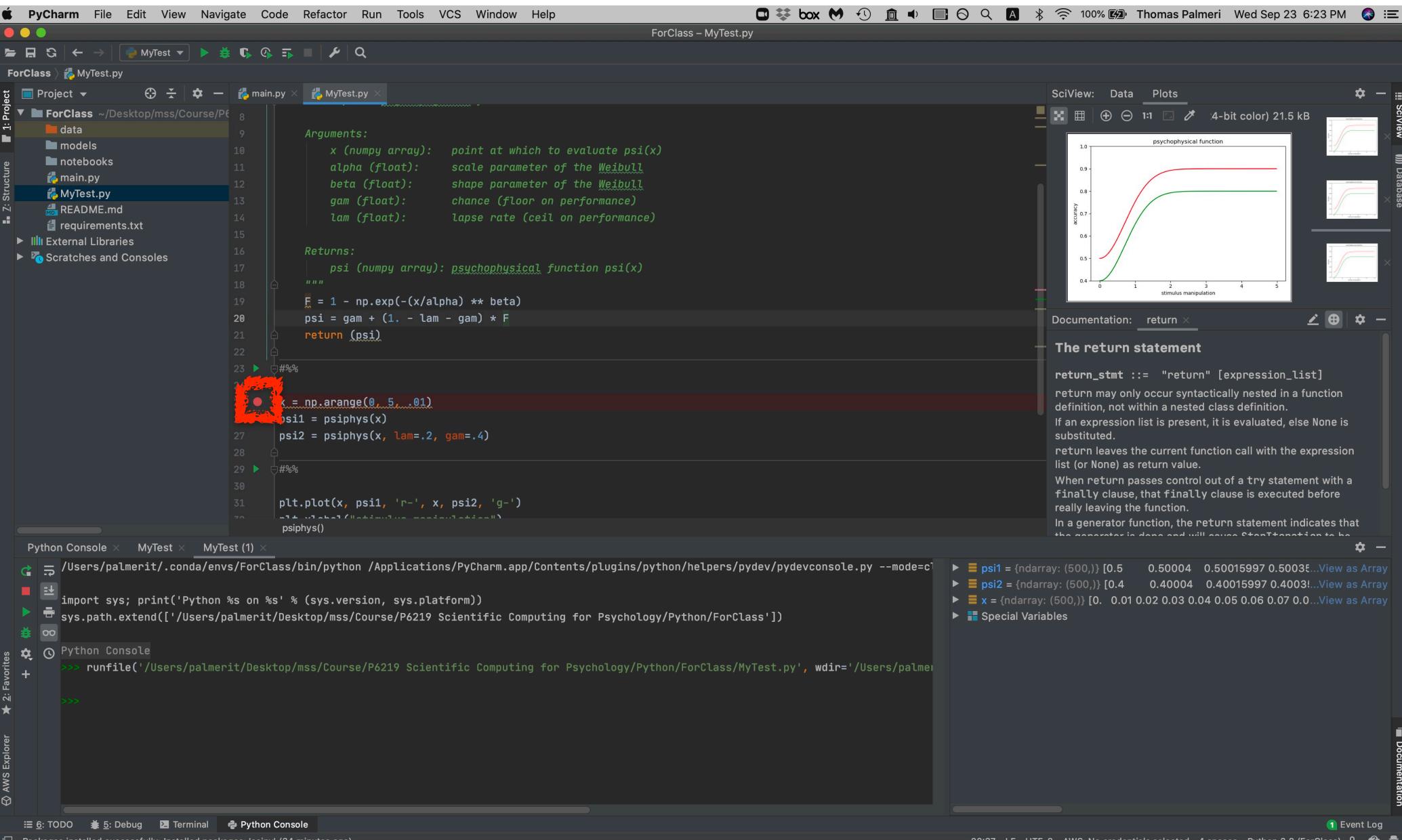
# debugging in PyCharm

Photo # NH 96566-KN (Color) First Computer "Bug", 1947



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)



The screenshot shows the PyCharm IDE interface with the following details:

- File Menu:** PyCharm, File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help.
- Toolbar:** Includes icons for file operations like Open, Save, and Run.
- Project View:** Shows the project structure under "ForClass".
- Code Editor:** The file "MyTest.py" is open. A red circle highlights the gutter area where a breakpoint is being set. The code defines a function "psiphys" with arguments x, alpha, beta, gamma, and lam, and returns a psychophysical function psi(x). It also contains a plot command.
- SciView:** A panel titled "SciView" displays a graph of "psychophysical function" accuracy versus stimulus manipulation. The graph shows two curves: a red curve starting at (0,0) and a green curve starting at approximately (0,0.4).
- Documentation:** A sidebar provides documentation for the "return" statement, including examples and syntax rules.
- Python Console:** The console output shows the execution of "runfile" command, which runs the script and displays the SciView plot.
- Bottom Status Bar:** Shows the current time (20:37), encoding (LF), file encoding (UTF-8), AWS status, number of spaces (4), Python version (3.8), and event log.

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

PyCharm File Edit View Navigate Code Refactor Run Tools VCS Window Help

ForClass – MyTest.py

ForClass MyTest.py Debug 'MyTest' ^D

Project 1: Project ForClass ~Desktop/mss/Course/P6219 Scientific Computing for Psychology/Python/ForClass

Structure 7: Structure

SciView: Data Plots 4-bit color 21.5 kB SciView Database

Arguments:

```
x (numpy array): point at which to evaluate psi(x)
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)
```

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 the generator is done and will cause StopIteration to be
```

PyCharm Console x MyTest x MyTest (1) x

```
/Users/palmerit/.conda/envs/ForClass/bin/python /Applications/PyCharm.app/Contents/plugins/python/helpers/pydev/pydevconsole.py --mode=cl
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.append(['/Users/palmerit/Desktop/mss/Course/P6219 Scientific Computing for Psychology/Python/ForClass'])

```

Python Console

```
>>> runfile('/Users/palmerit/Desktop/mss/Course/P6219 Scientific Computing for Psychology/Python/ForClass/MyTest.py', wdir='/Users/palmerit/Desktop/mss/Course/P6219 Scientific Computing for Psychology/Python/ForClass')
```

Variables

- psi1 = {ndarray: (500,)} [0.5 0.50004 0.50015997 0.5003...View as Array
- psi2 = {ndarray: (500,)} [0.4 0.40004 0.40015997 0.4003!...View as Array
- x = {ndarray: (500,)} [0. 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.0...View as Array
- Special Variables

AWS Explorer

Event Log

Debug selected configuration

20:37 LF UTF-8 AWS: No credentials selected 4 spaces Python 3.8 (ForClass)

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

Screenshot of PyCharm IDE showing a Python script named MyTest.py. The code defines a function psi and uses it to plot two psychophysical functions.

```
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, lam=.2, gam=.4)
```

The line `X = np.arange(0, 5, .01)` is highlighted with a red box and is the current line of execution, indicated by a red dot in the gutter. The SciView panel shows two plots: one for the psychophysical function and another for the accuracy.

SciView: Data Plots  
psychophysical function  
accuracy  
stimulus manipulation

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  
Frames  
MainThread  
<module>, MyTest.py:25

Variables

AWS Explorer  
Favorites

Event Log

Packages installed successfully: Installed packages: 'scipy' (45 minutes ago)

PyCharm File Edit View Navigate Code Refactor Run Tools VCS Window Help

ForClass – MyTest.py

ForClass > MyTest.py

Project 1: Project ~Desktop/mss/Course/P6 17

Structure

1: Project

ForClass

- data
- models
- notebooks
- main.py
- MyTest.py
- README.md
- requirements.txt

External Libraries

Scratches and Consoles

SciView: Data Plots 4-bit color 21.5 kB

psychophysical function

accuracy

stimulus manipulation

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.

Debug: MyTest

Debugger

Frames

MainThread

<module>, MyTest.py:25

Variables

Special Variables

AWS Explorer

TODO Debug Terminal Python Console Event Log

This screenshot shows the PyCharm IDE interface. The top navigation bar includes PyCharm, File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The title bar says 'ForClass – MyTest.py'. The left sidebar has a 'Project' view with a tree structure for the 'ForClass' project, including subfolders like 'data', 'models', 'notebooks', 'main.py', 'MyTest.py', 'README.md', and 'requirements.txt'. Below this is an 'External Libraries' section and a 'Scratches and Consoles' section. The main code editor window shows 'MyTest.py' with code related to a psychophysical function. A red highlight is on line 25: 'x = np.arange(0, 5, .01)'. To the right of the code editor is a 'SciView' panel titled 'psychophysical function' showing three plots of accuracy vs stimulus manipulation. The bottom part of the interface is the 'Debug' toolbar, which is highlighted with a red border. It includes tabs for Debugger, Console, and Frames, with 'Debugger' selected. The 'Frames' tab shows the 'MainThread' frame and the line '<module>, MyTest.py:25'. The 'Variables' tab is also visible. The bottom status bar shows 'TODO', 'Debug', 'Terminal', 'Python Console', and 'Event Log'.

notice that the bottom has switched into Debug mode

PyCharm File Edit View Navigate Code Refactor Run Tools VCS Window Help

ForClass – MyTest.py

ForClass > MyTest.py

Project 1: Project ~Desktop/mss/Course/P6 17

1: Structure

1: SciView: Data Plots

1: SciView

1: Database

1: Documentation

1: The return statement

1: return\_stmt ::= "return" [expression\_list]

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

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

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

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

1: In a generator function, the return statement indicates that

1: Documentation

1: Psi (numpy array): psychophysical function psi(x)

1: """

1: F = 1 - np.exp(-(x/alpha) \*\* beta)

1: psi = gam + (1. - lam - gam) \* F

1: return (psi)

1: %%

1: X = np.arange(0, 5, .01)

1: psi1 = psiphys(X)

1: psi2 = psiphys(X, lam=.2, gam=.4)

1: %%

1: plt.plot(X, psi1, 'r-', X, psi2, 'g-')

1: plt.xlabel("stimulus manipulation")

1: plt.ylabel("accuracy")

1: plt.title("psychophysical function")

1: plt.ylim(.4, 1)

1: plt.show()

1: Documentation: return

1: The return statement

1: return\_stmt ::= "return" [expression\_list]

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

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

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

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

1: In a generator function, the return statement indicates that

1: Documentation

Debug: MyTest

Debugger Console

Frames

MainThread <module>, MyTest.py:25

Variables

2: Favorites

2: AWS Explorer

2: Terminal

2: Python Console

2: Event Log

2: Packages installed successfully: 'scipy' (45 minutes ago)

25:1 LF UTF-8 AWS: No credentials selected 4 spaces Python 3.8 (ForClass)

can still access the Terminal

PyCharm File Edit View Navigate Code Refactor Run Tools VCS Window Help

ForClass – MyTest.py

ForClass > MyTest.py

Project 1: Project ~Desktop/mss/Course/P6 17

1: Structure

1: SciView: Data Plots

1: SciView

1: Database

1: Documentation

1: The return statement

1: return\_stmt ::= "return" [expression\_list]

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

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

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

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

1: In a generator function, the return statement indicates that

```
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")
plt.ylim(.4, 1)
plt.show()
```

Debug: MyTest

Debugger Console

Frames MainThread <module>, MyTest.py:25

Variables

2: Favorites

2: AWS Explorer

2: Documentation

6: TODO 5: Debug 4: Terminal Python Console

Packages installed successfully: Installed pack... (2 days ago)

Event Log

25:1 LF UTF-8 AWS: No credentials selected 4 spaces Python 3.8 (ForClass)

and the independent Python Console

PyCharm File Edit View Navigate Code Refactor Run Tools VCS Window Help

ForClass – MyTest.py

ForClass > MyTest.py

Project 1: Project ~Desktop/mss/Course/P6 17

1: Structure

1: SciView: Data Plots

1: SciView

1: Database

1: Documentation

1: The return statement

1: return\_stmt ::= "return" [expression\_list]

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

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

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

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

1: In a generator function, the return statement indicates that

```
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")
plt.ylim(.4, 1)
plt.show()
```

Debug: MyTest

Debugger Console

Frames

MainThread

<module>, MyTest.py:25

Variables

Special Variables

2: Favorites

2: AWS Explorer

2: Documentation

6: TODO 5: Debug 4: Terminal Python Console

Packages installed successfully: Installed packages: 'scipy' (45 minutes ago)

25:1 LF UTF-8 AWS: No credentials selected 4 spaces Python 3.8 (ForClass) Event Log

The screenshot shows the PyCharm IDE interface. The top navigation bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The main window displays a project named 'ForClass' with files like 'main.py' and 'MyTest.py'. The code editor shows Python code for calculating psychophysical functions and plotting them. A SciView panel on the right displays a plot titled 'psychophysical function' showing accuracy versus stimulus manipulation for two conditions. The Debug panel at the bottom shows the current stack trace, and the AWS Explorer panel shows favorite AWS services. The bottom status bar indicates the file is 25:1, uses LF line endings, is in UTF-8 encoding, and is using Python 3.8.

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 Structure

1: Project ForClass ~/Desktop/mss/Course/ForClass

2: Structure

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

psychophysical function

accuracy

stimulus manipulation

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 "/Users/palmerit/PycharmProjects/ForClass/MyTest.py"
```

pydev debugger: process 85316 is connecting

Connected to pydev debugger (build 201.8743.11)

AWS Explorer

2: Favorites

Event Log

TODO Debug Terminal Python Console

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

PyCharm File Edit View Navigate Code Refactor Run Tools VCS Window Help

ForClass – MyTest.py

ForClass > MyTest.py

Project 1: Project ~Desktop/mss/Course/P6 17

Structure

1: Project ForClass ~Desktop/mss/Course/P6 17

data

models

notebooks

main.py

MyTest.py

README.md

requirements.txt

External Libraries

Scratches and Consoles

SciView: Data Plots 4-bit color 21.5 kB

psychophysical function

accuracy

stimulus manipulation

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

Frames MainThread

<module>, MyTest.py:25

Variables

AVS Explorer

TODO Debug Terminal Python Console

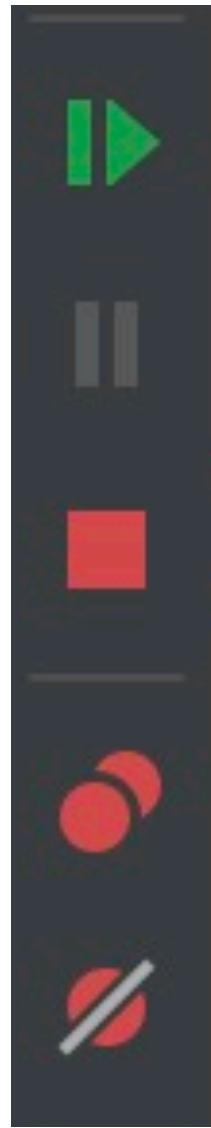
Packages installed successfully: Installed packages: 'scipy' (45 minutes ago)

Event Log

25:1 LF UTF-8 AWS: No credentials selected 4 spaces Python 3.8 (ForClass)

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

resume  
program



pause  
program



stop  
program



view  
breakpoints



mute  
breakpoints



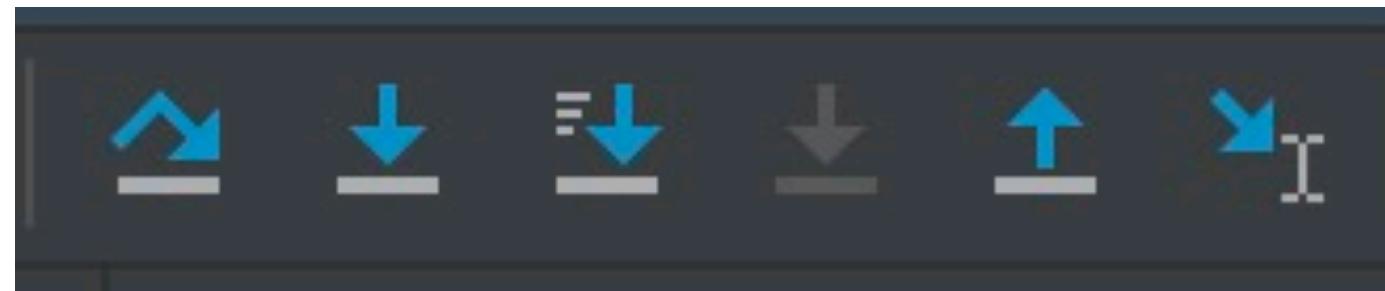
step  
over

step  
into

step  
into  
*my code*

step  
out

run  
to  
cursor



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

The screenshot shows the PyCharm IDE interface with the following components visible:

- Top Bar:** PyCharm, File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help.
- Toolbar:** Standard file operations (New, Open, Save, Find, etc.).
- Project View:** Shows the project structure with a folder named "ForClass" containing files like "main.py", "MyTest.py", and "README.md".
- Code Editor:** The "MyTest.py" file is open, showing Python code for calculating a psychophysical function and plotting it. A red box highlights the line of code that generates the plot data: `x = np.arange(0, 5, .01) x: [0. 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.1 0.11 0.12 0.13, 0.14 0.15 0.16 0.17]`. The plot window on the right displays two sigmoidal curves labeled "psi1" and "psi2" with the title "psychophysical function".
- SciView:** Data and Plots tab. The Data tab shows a table of accuracy values corresponding to the x-axis points. The Plots tab shows three additional sigmoidal curves.
- Documentation:** Shows the documentation for the `return` statement.
- Debug View:** Shows the current stack frame and variable values. A red box highlights the "Variables" section, which lists `psi1`, `psi2`, and `x` as arrays, along with "Special Variables".
- Bottom Status Bar:** Shows the file path "ForClass - MyTest.py", line count "25:1", encoding "LF", character set "UTF-8", AWS status, and Python version "Python 3.8 (ForClass)".

PyCharm File Edit View Navigate Code Refactor Run Tools VCS Window Help

Thomas Palmeri Wed Sep 23 7:51 PM

ForClass – MyTest.py

Project: ForClass ~/Desktop/mss/Course/Python/ForClass

Structure: data, models, notebooks, main.py, MyTest.py, README.md, requirements.txt, External Libraries, Scratches and Consoles

SciView: Data Plots 4-bit color 21.5 kB

ForClass > MyTest.py

main.py MyTest.py

def psiphys(x, alpha=1., beta=2., gam=.5, lam=.1):  
 """Computes psychophysical function  
 Arguments:  
 x (numpy array): point at which to evaluate psi(x)  
 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)

psiphys()

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

Variables

MainThread

psiphys, MyTest.py:19

<module>, MyTest.py:26

alpha = {float} 1.0  
beta = {float} 2.0  
gam = {float} 0.5  
lam = {float} 0.1

x = {ndarray: (500,)} [0. 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.1 0.11 0.12 0.13 0.14 0.15 0.16 0.17 0.18 0.19 0.2 0.21 0.22 0.23 0.24 0.25 0.26 0.27 0.28 0.29 0.3 0.31 0.32 0.33 0.34 0.35 0.36 0.37 ...View as Array

Event Log

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

25:1 LF UTF-8 AWS: No credentials selected 4 spaces Python 3.8 (ForClass)

stepping into a function, you see the local variables

PyCharm File Edit View Navigate Code Refactor Run Tools VCS Window Help

ForClass – MyTest.py

Project: ForClass ~/Desktop/mss/Course/ForClass

Structure: 1: Project ForClass data notebooks main.py MyTest.py README.md requirements.txt External Libraries Scratches and Consoles

SciView: Data Plots 4-bit color 21.5 kB SciView Database

ForClass > MyTest.py

```

def psiphys(x, alpha=1., beta=2., gam=.5, lam=.1):
    """Computes psychophysical function
    Arguments:
        x (numpy array): point at which to evaluate psi(x)
        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

```

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

Variables:

- MainThread
- psiphys, MyTest.py:19
- <module>, MyTest.py:26
- x = [ndarray: (500,)] [0. 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.1 0.11 0.12 0.13, 0.14 0.15 0.16 0.17 0.18 0.19 0.2 0.21 0.22 0.23 0.24 0.25 0.26 0.27, 0.28 0.29 0.3 0.31 0.32 0.33 0.34 0.35 0.36 0.37 ...View as Array

AWS Explorer

Event Log

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

25:1 LF UTF-8 AWS: No credentials selected 4 spaces Python 3.8 (ForClass)

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

PyCharm File Edit View Navigate Code Refactor Run Tools VCS Window Help

ForClass – MyTest.py

Project: ForClass ~/Desktop/mss/Course/ForClass

Structure: 1: Project, 2: Structure

SciView: Data Plots

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

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")
psiphys()
```

Debug: MyTest

Debugger Console

```
/Users/palmeri/.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 56427 --file "/Users/palmeri/PycharmProjects/ForClass/MyTest.py"
```

Connected to pydev debugger (build 201.8743.11)

>>> alpha  
1.0  
>>> alpha = 5  
>>> alpha  
5  
>>> |

Todo: 6: TODO

Debug: 5: Debug

Terminal

Python Console

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

Event Log

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

ForClass – MyTest.py

ForClass > MyTest.py

Project: ForClass ~/Desktop/mss/Course/Python/ForClass

SciView: Data Plots

SciView Database

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

```

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")
psiphys()

```

Debug: MyTest

Debugger Console

Frames Variables

Connected

Frames are not available

AWS Explorer Favorites

Event Log

6: TODO 5: Debug 4: Terminal Python Console

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

19:1 LF UTF-8 AWS: No credentials selected 4 spaces Python 3.8 (ForClass)

PyCharm File Edit View Navigate Code Refactor Run Tools VCS Window Help

ForClass – MyTest.py

ForClass > MyTest.py

Project

1: Project ForClass ~/Desktop/mss/Course/P6219 Scientific Computing for Psychology/Python/ForClass

2: Structure

3: SciView: Data Plots 4-bit color 21.5 kB

4: SciView Database

5: Documentation

6: Python Console

7: Favorites

8: AWS Explorer

9: Documentation

10: Event Log

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

12: 25:1 LF UTF-8 AWS: No credentials selected 4 spaces Python 3.8 (ForClass)

13: ForClass – MyTest.py

14: import matplotlib.pyplot as plt

15: #%

16: def psiphys(x, alpha=1., beta=2., gam=.5, lam=.1):

17: """Computes psychophysical function

18:

19: Arguments:

20: x (numpy array): point at which to evaluate psi(x)

21: alpha (float): scale parameter of the Weibull

22: beta (float): shape parameter of the Weibull

23: gam (float): chance (floor on performance)

24: lam (float): lapse rate (ceil on performance)

25: Returns:

26: psi (numpy array): psychophysical function psi(x)

27: """

28: F = 1 - np.exp(-(x/alpha) \*\* beta)

29: psi = gam + (1. - lam - gam) \* F

30: return (psi)

31: #%

32: x = np.arange(0, 5, .01)

33: print(psiphys(x))

34: #%

35: #

36: #

37: #

38: #

39: #

40: #

41: #

42: #

43: #

44: #

45: #

46: #

47: #

48: #

49: #

50: #

51: #

52: #

53: #

54: #

55: #

56: #

57: #

58: #

59: #

60: #

61: #

62: #

63: #

64: #

65: #

66: #

67: #

68: #

69: #

70: #

71: #

72: #

73: #

74: #

75: #

76: #

77: #

78: #

79: #

80: #

81: #

82: #

83: #

84: #

85: #

86: #

87: #

88: #

89: #

90: #

91: #

92: #

93: #

94: #

95: #

96: #

97: #

98: #

99: #

100: #

101: #

102: #

103: #

104: #

105: #

106: #

107: #

108: #

109: #

110: #

111: #

112: #

113: #

114: #

115: #

116: #

117: #

118: #

119: #

120: #

121: #

122: #

123: #

124: #

125: #

126: #

127: #

128: #

129: #

130: #

131: #

132: #

133: #

134: #

135: #

136: #

137: #

138: #

139: #

140: #

141: #

142: #

143: #

144: #

145: #

146: #

147: #

148: #

149: #

150: #

151: #

152: #

153: #

154: #

155: #

156: #

157: #

158: #

159: #

160: #

161: #

162: #

163: #

164: #

165: #

166: #

167: #

168: #

169: #

170: #

171: #

172: #

173: #

174: #

175: #

176: #

177: #

178: #

179: #

180: #

181: #

182: #

183: #

184: #

185: #

186: #

187: #

188: #

189: #

190: #

191: #

192: #

193: #

194: #

195: #

196: #

197: #

198: #

199: #

200: #

201: #

202: #

203: #

204: #

205: #

206: #

207: #

208: #

209: #

210: #

211: #

212: #

213: #

214: #

215: #

216: #

217: #

218: #

219: #

220: #

221: #

222: #

223: #

224: #

225: #

226: #

227: #

228: #

229: #

230: #

231: #

232: #

233: #

234: #

235: #

236: #

237: #

238: #

239: #

240: #

241: #

242: #

243: #

244: #

245: #

246: #

247: #

248: #

249: #

250: #

251: #

252: #

253: #

254: #

255: #

256: #

257: #

258: #

259: #

260: #

261: #

262: #

263: #

264: #

265: #

266: #

267: #

268: #

269: #

270: #

271: #

272: #

273: #

274: #

275: #

276: #

277: #

278: #

279: #

280: #

281: #

282: #

283: #

284: #

285: #

286: #

287: #

288: #

289: #

290: #

291: #

292: #

293: #

294: #

295: #

296: #

297: #

298: #

299: #

300: #

301: #

302: #

303: #

304: #

305: #

306: #

307: #

308: #

309: #

310: #

311: #

312: #

313: #

314: #

315: #

316: #

317: #

318: #

319: #

320: #

321: #

322: #

323: #

324: #

325: #

326: #

327: #

328: #

329: #

330: #

331: #

332: #

333: #

334: #

335: #

336: #

337: #

338: #

339: #

340: #

341: #

342: #

343: #

344: #

345: #

346: #

347: #

348: #

349: #

350: #

351: #

352: #

353: #

354: #

355: #

356: #

357: #

358: #

359: #

360: #

361: #

362: #

363: #

364: #

365: #

366: #

367: #

368: #

369: #

370: #

371: #

372: #

373: #

374: #

375: #

376: #

377: #

378: #

379: #

380: #

381: #

382: #

383: #

384: #

385: #

386: #

387: #

388: #

389: #

390: #

391: #

392: #

393: #

394: #

395: #

396: #

397: #

398: #

399: #

400: #

401: #

402: #

403: #

404: #

405: #

406: #

407: #

408: #

409: #

410: #

411: #

412: #

413: #

414: #

415: #

416: #

417: #

418: #

419: #

420: #

421: #

422: #

423: #

424: #

425: #

426: #

427: #

428: #

429: #

430: #

431: #

432: #

433: #

434: #

435: #

436: #

437: #

438: #

439: #

440: #

441: #

442: #

443: #

444: #

445: #

446: #

447: #

448: #

449: #

450: #

451: #

452: #

453: #

454: #

455: #

456: #

457: #

458: #

459: #

460: #

461: #

462: #

463: #

464: #

465: #

466: #

467: #

468: #

469: #

470: #

471: #

472: #

473: #

474: #

475: #

476: #

477: #

478: #

479: #

480: #

481: #

482: #

483: #

484: #

485: #

486: #

487: #

488: #

489: #

490: #

491: #

492: #

493: #

494: #

495: #

496: #

497: #

498: #

499: #

500: #

501: #

502: #

503: #

504: #

505: #

506: #

507: #

508: #

509: #

510: #

511: #

512: #

513: #

514: #

515: #

516: #

517: #

518: #

519: #

520: #

521: #

522: #

523: #

524: #

525: #

526: #

527: #

528: #

529: #

530: #

531: #

532: #

533: #

534: #

535: #

536: #

537: #

538: #

539: #

540: #

541: #

542: #

543: #

544: #

545: #

546: #

547: #

548: #

549: #

550: #

551: #

552: #

553: #

554: #

555: #

556: #

557: #

558: #

559: #

560: #

561: #

562: #

563: #

564: #

565: #

566: #

567: #

568: #

569: #

570: #

571: #

572: #

573: #

574: #

575: #

576: #

577: #

578: #

579: #

580: #

581: #

582: #

583: #

584: #

585: #

586: #

587: #

588: #

589: #

590: #

591: #

592: #

593: #

594: #

595: #

596: #

597: #

598: #

599: #

600: #

601: #

602: #

603: #

604: #

605: #

606: #

607: #

608: #

609: #

610: #

611: #

612: #

613: #

614: #

615: #

616: #

617: #

618: #

619: #

620: #

621: #

622: #

623: #

624: #

625: #

626: #

627: #

628: #

629: #

630: #

631: #

632: #

633: #

634: #

635: #

636: #

637: #

638: #

639: #

640: #

641: #

642: #

643: #

644: #

645: #

646: #

647: #

648: #

649: #

650: #

651: #

652: #

653: #

654: #

655: #

656: #

657: #

658: #

659: #

660: #

661: #

662: #

663: #

664: #

665: #

666: #

667: #

668: #

669: #

670: #

671: #

672: #

673: #

674: #

675: #

676: #

677: #

678: #

679: #

680: #

681: #

682: #

683: #

684: #

685: #

686: #

687: #

688: #

689: #

690: #

691: #

692: #

693: #

694: #

695: #

696: #

697: #

698: #

699: #

700: #

701: #

702: #

703: #

704: #

705: #

706: #

707: #

708: #

709: #

710: #

711: #

712: #

713: #

714: #

715: #

716: #

717: #

718: #

719: #

720: #

721: #

722: #

723: #

724: #

725: #

726: #

727: #

728: #

729: #

730: #

731: #

732: #

733: #

734: #

735: #

736: #

737: #

738: #

739: #

740: #

741: #

742: #

743: #

744: #

745: #

746: #

747: #

748: #

749: #

750: #

751: #

752: #

753: #

754: #

755: #

756: #

757: #

758: #

759: #

760: #

761: #

762: #

763: #

764: #

765: #

766: #

767: #

768: #

769: #

770: #

771: #

772: #

773: #

774: #

775: #

776: #

777: #

778: #

779: #

780: #

781: #

782: #

783: #

784: #

785: #

786: #

787: #

788: #

789: #

790: #

791: #

792: #

793: #

794: #

795: #

796: #

797: #

798: #

799: #

800: #

801: #

802: #

803: #

804: #

805: #

806: #

807: #

808: #

809: #

810: #

811: #

812: #

813: #

814: #

815: #

816: #

817: #

818: #

819: #

820: #

821: #

822: #

823: #

824: #

825: #

826: #

827: #

828: #

829: #

830: #

831: #

832: #

833: #

834: #

835: #

836: #

837: #

838: #

839: #

840: #

841: #

842: #

843: #

844: #

845: #

846: #

847: #

848: #

849: #

850: #

851: #

852: #

853: #

854: #

855: #

856: #

857: #

858: #

859: #

860: #

861: #

862: #

863: #

864: #

865: #

866: #

867: #

868: #

869: #

870: #

871: #

872: #

873: #

874: #

875: #

876: #

877: #

878: #

879: #

880: #

881: #

882: #

883: #

884: #

885: #

886: #

887: #

888: #

889: #

890: #

891: #

892: #

893: #

894: #

895: #

896: #

897: #

898: #

899: #

900: #

901: #

902: #

903: #

904: #

905: #

906: #

907: #

908: #

909: #

910: #

911: #

912: #

913: #

914: #

915: #

916: #

917: #

918: #

919: #

920: #

921: #

922: #

923: #

924: #

925: #

926: #

927: #

928: #

929: #

930: #

931: #

932: #

933: #

934: #

935: #

936: #

937: #

938: #

939: #

940: #

941: #

942: #

943: #

944: #

945: #

946: #

947: #

948: #

949: #

950: #

951: #

952: #

953: #

954: #

955: #

956: #

957: #

958: #

959: #

960: #

961: #

962: #

963: #

964: #

965: #

966: #

967: #

968: #

969: #

970: #

971: #

972: #

973: #

974: #

975: #

976: #

977: #

978: #

979: #

980: #

981: #

982: #

983: #

984: #

985: #

986: #

987: #

988: #

989: #

990: #

991: #

992: #

993: #

994: #

995: #

996: #

997: #

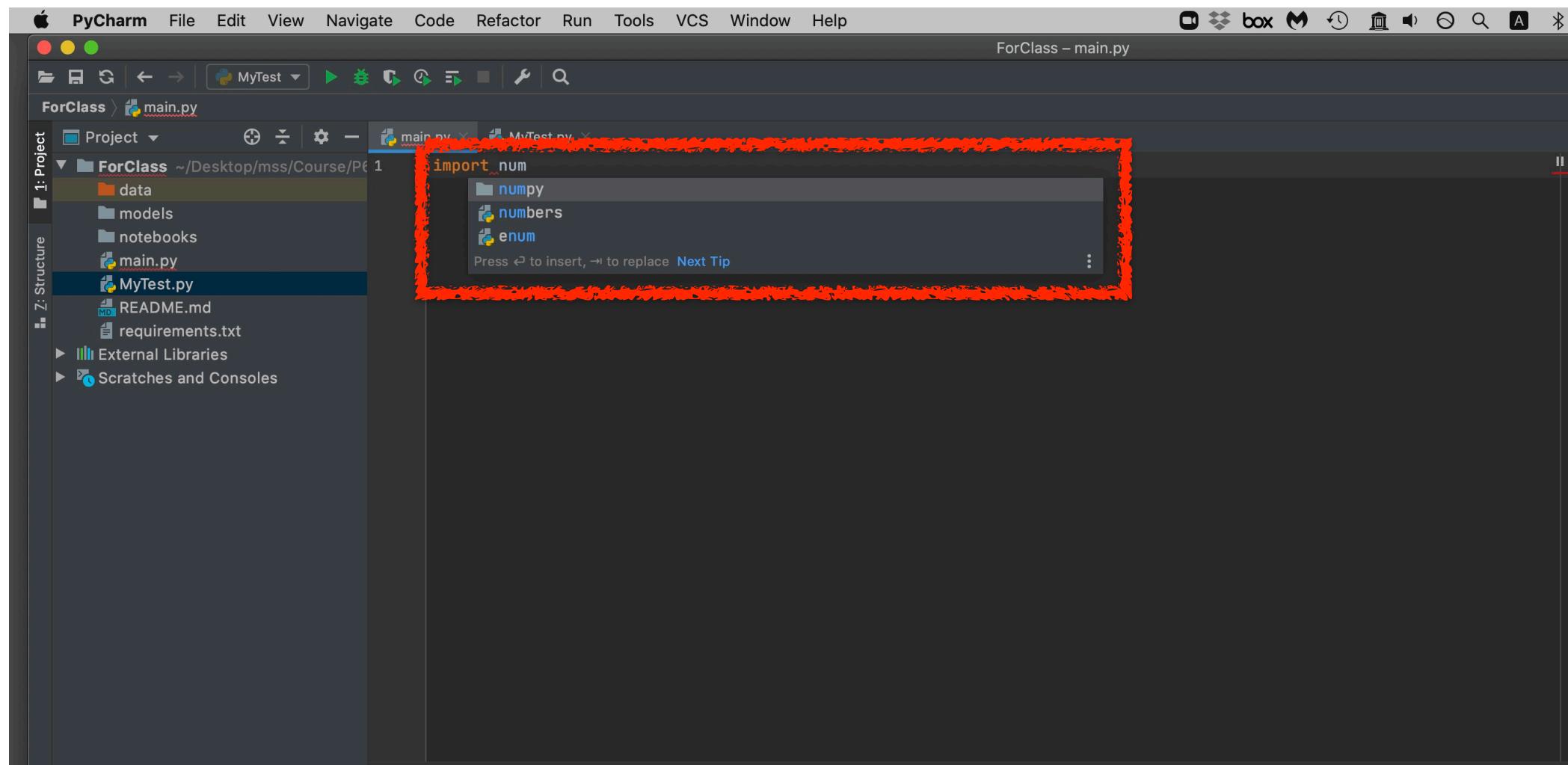
998: #

999: #

1000: #

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

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



The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The title bar indicates the file is "ForClass – main.py". The left sidebar has sections for Project, Structure, and Scratches and Consoles. The main editor window displays Python code for a function named "psiphys". A red box highlights the line "f psiphys(x, alpha, beta, gam, lam)". A tooltip below this line provides documentation for the function, listing arguments and their descriptions. The code uses standard Python syntax with imports for numpy and matplotlib.pyplot.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def psiphys(x, alpha=1., beta=2., gam=.5, lam=.1):
5     """Computes psychophysical function
6
7     Arguments:
8         x (numpy array): point at which to evaluate psi(x)
9         alpha (float): scale parameter of the Weibull
10        beta (float): shape parameter of the Weibull
11        gam (float): chance (floor on performance)
12        lam (float): lapse rate (ceil on performance)
13
14     Returns:
15         psi (numpy array): psychophysical function psi(x)
16     """
17     F = 1 - np.exp(-(x/alpha) ** beta)
18     psi = gam + (1. - lam - gam) * F
19     return (psi)
20
21 x = np.arange(0, 5, .01)
22 psi1 = psiphys
23 f psiphys(x, alpha, beta, gam, lam)
```

autocomplete 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 > main.py

Project main.py MyTest.py

1: Project 1: ForClass ~~/Desktop/mss/Course/P 1  
2: data  
3: models  
4: notebooks  
5: main.py  
6: MyTest.py  
7: README.md  
8: requirements.txt

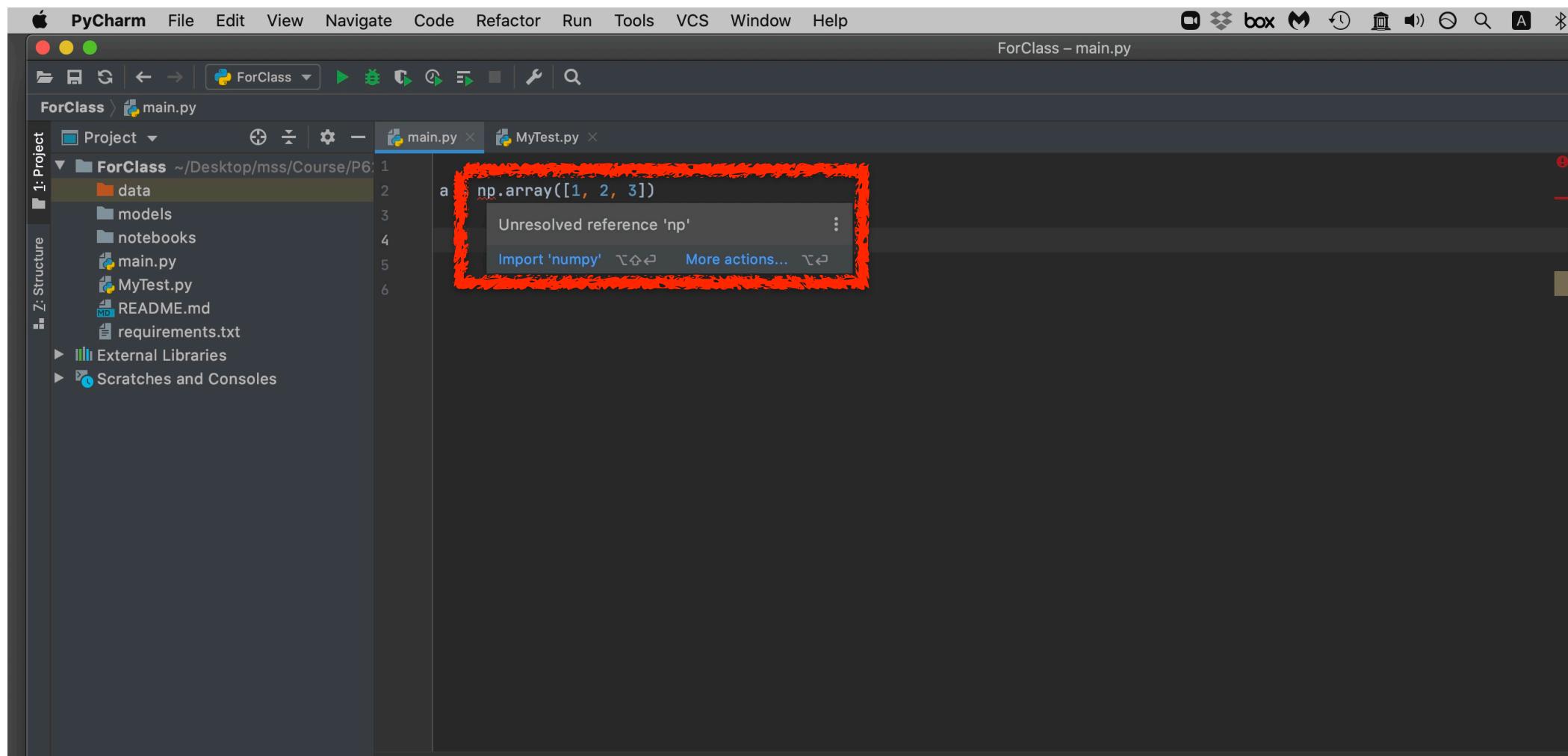
External Libraries  
Scratches and Consoles

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

v arr  
bytearray  
builtins  
Press ⌘ to insert, ⌘ to replace Next Tip

The screenshot shows the PyCharm IDE interface. The top menu bar includes Apple logo, PyCharm, File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The title bar says "ForClass – main.py". Below the menu is a toolbar with various icons. The left sidebar has sections for "Project" and "Structure". The "Project" section shows a folder named "ForClass" containing subfolders "data", "models", "notebooks", and files "main.py" and "MyTest.py", along with "README.md" and "requirements.txt". The "Structure" section shows "External Libraries" and "Scratches and Consoles". The main editor area contains Python code: "import numpy as np", "arr = np.array([1, 2, 3, 4, 5])", and "print(arr)". A tooltip is displayed over the line "print(arr)" with the text "v arr", "bytearray", and "builtins". A red box highlights this tooltip. At the bottom of the tooltip, it says "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)

The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The title bar indicates the file is "ForClass – main.py". The left sidebar has sections for Project, Z: Structure, External Libraries, and Scratches and Consoles. The Project section shows a directory structure for "ForClass" containing "data", "models", "notebooks", "main.py", "MyTest.py", "README.md", and "requirements.txt". The main code editor window displays Python code. Line 4 contains the line "b = math.sin(2.4)". A red rectangular box highlights this line. A tooltip appears over the "math" reference, stating "Unresolved reference 'math'" with options to "Import 'math'" or "More actions...".

```
a = np.array([1, 2, 3])
b = math.sin(2.4)
Unresolved reference 'math'
Import 'math' More actions...
```



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__':
```

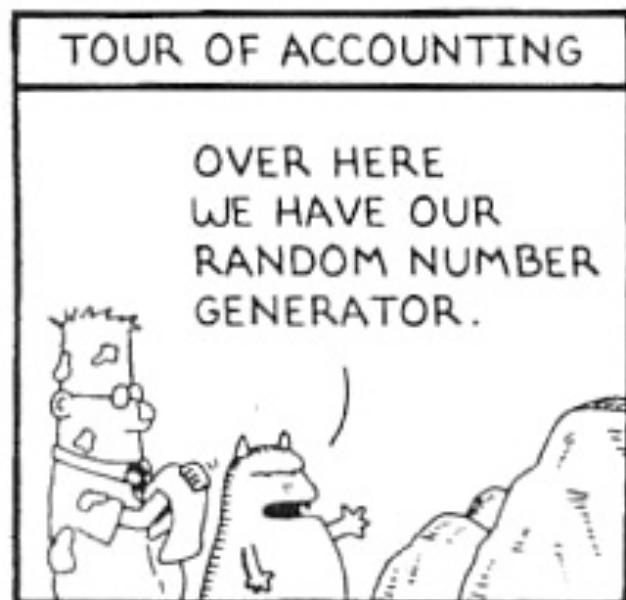
*things to run to test/demo the module*

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



# Random Numbers

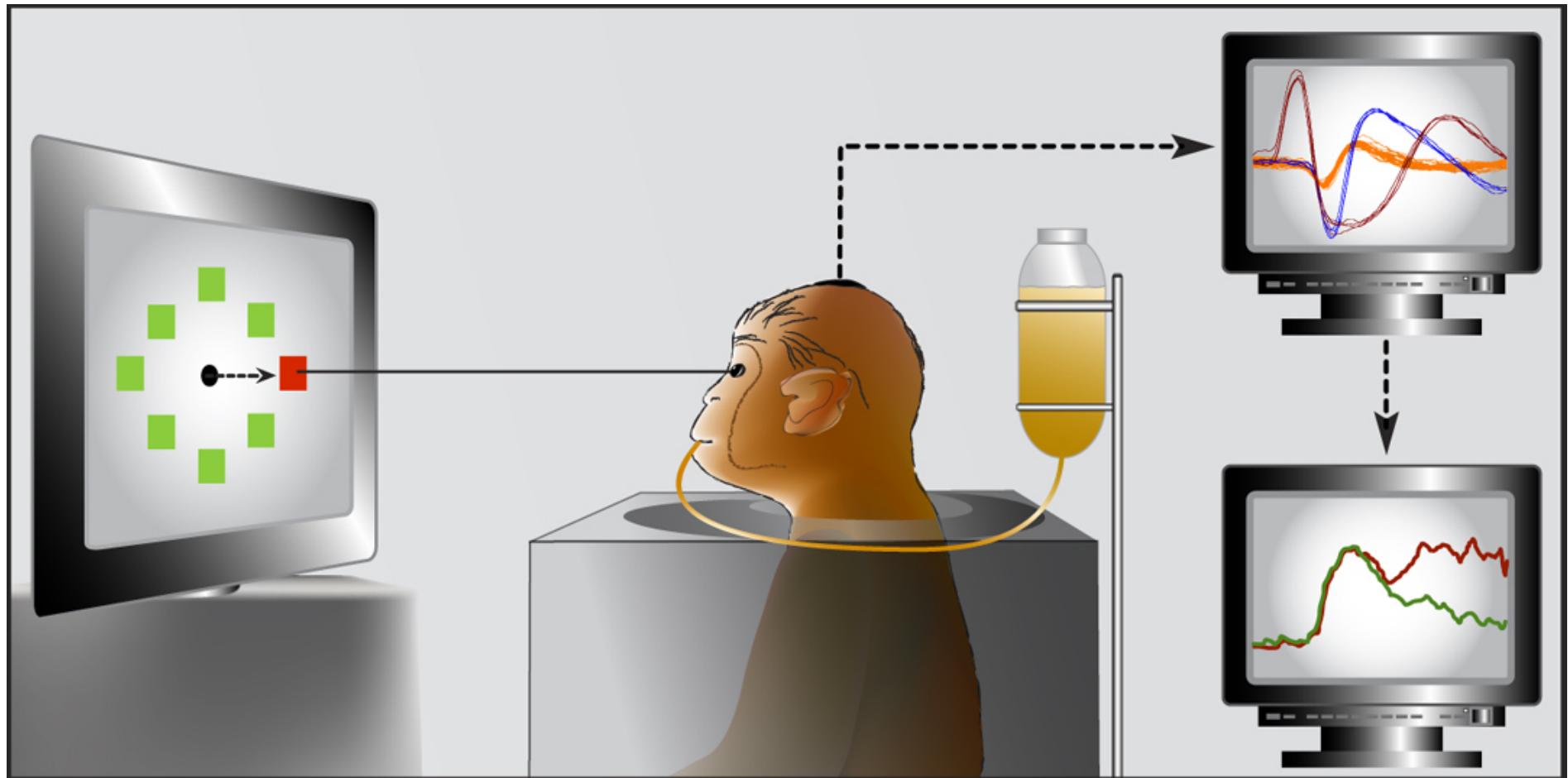
**DILBERT** By SCOTT ADAMS



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

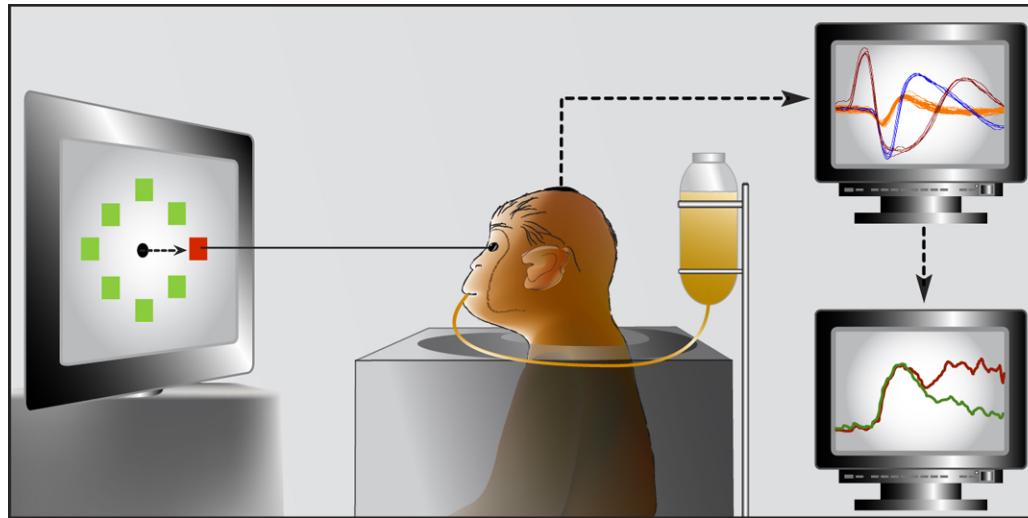
# uses of random numbers

(in psychology and neuroscience)



an example

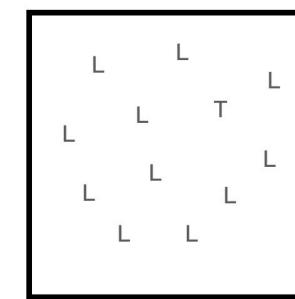
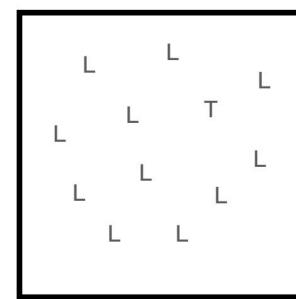
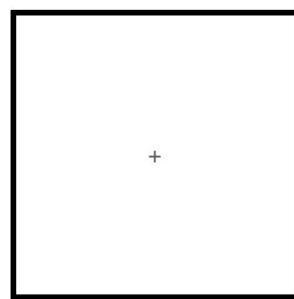
# randomization in a behavioral experiment



Trial 1

target present condition

correct

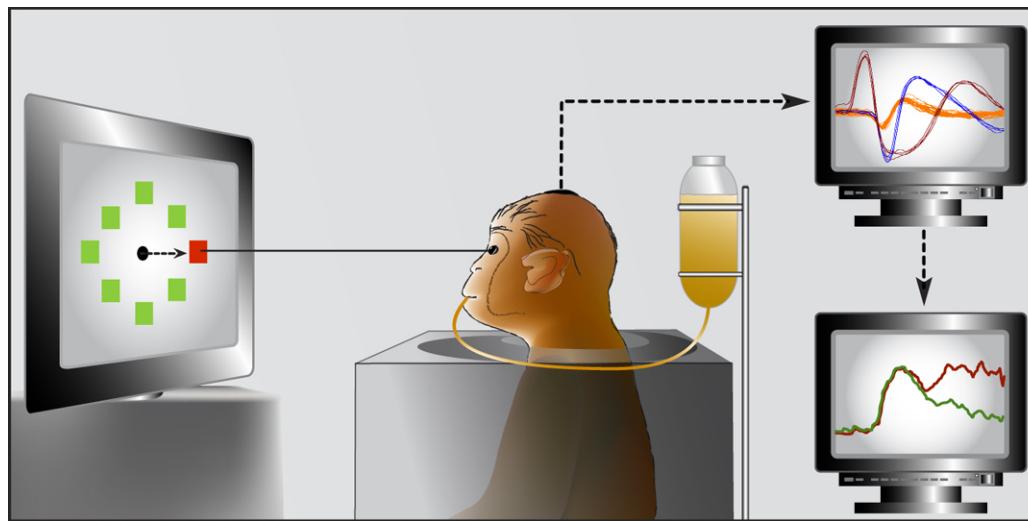


"present"

response time (RT)

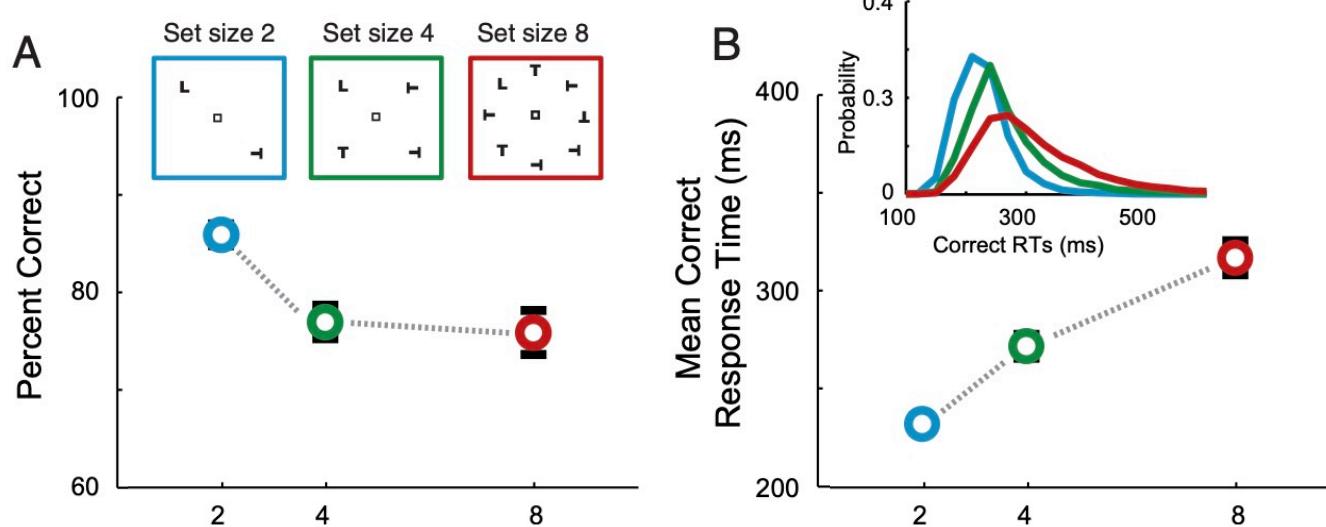
time

# randomization in a behavioral experiment



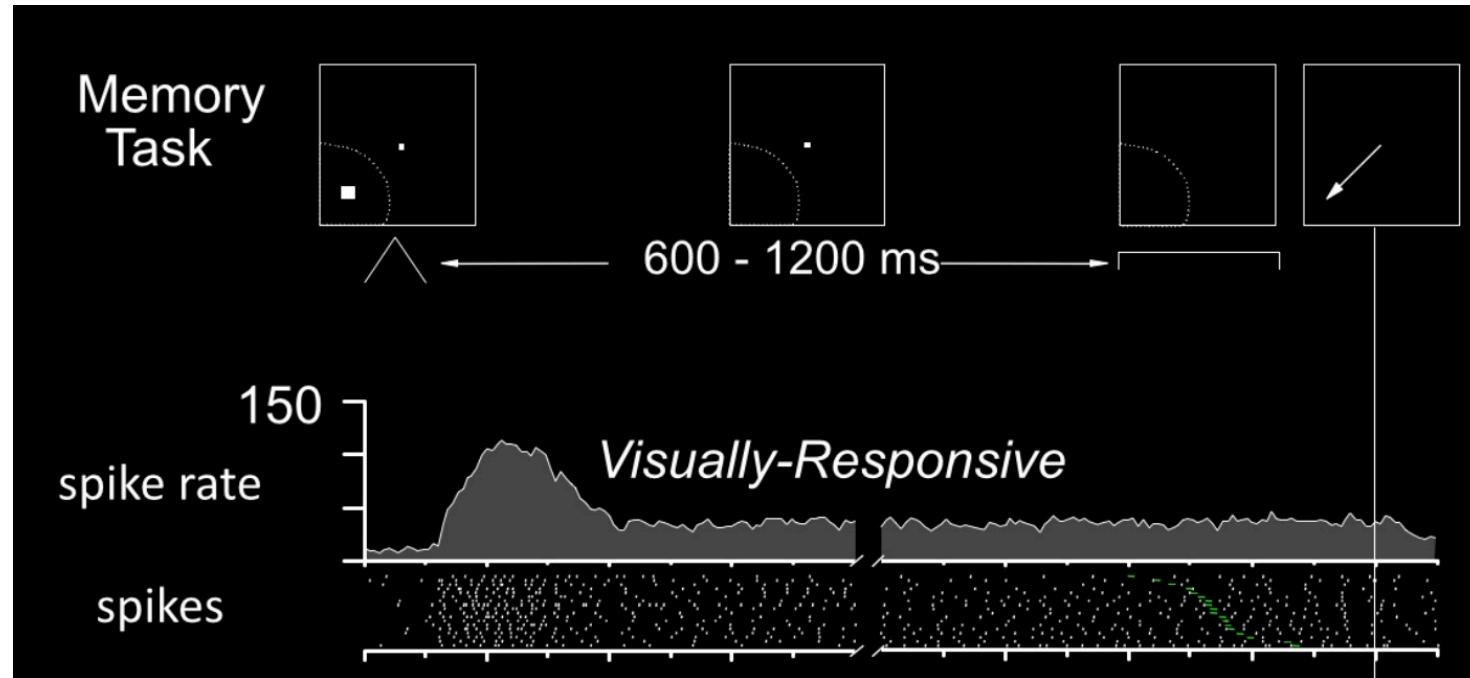
- randomize the presence/absence and the location of target and distractors in the array from trial to trial, randomize the set size from trial to trial
  - makes target location and search conditions unpredictable
- constrain to avoid repetitions of the same location from one trial to the next
  - breaking 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



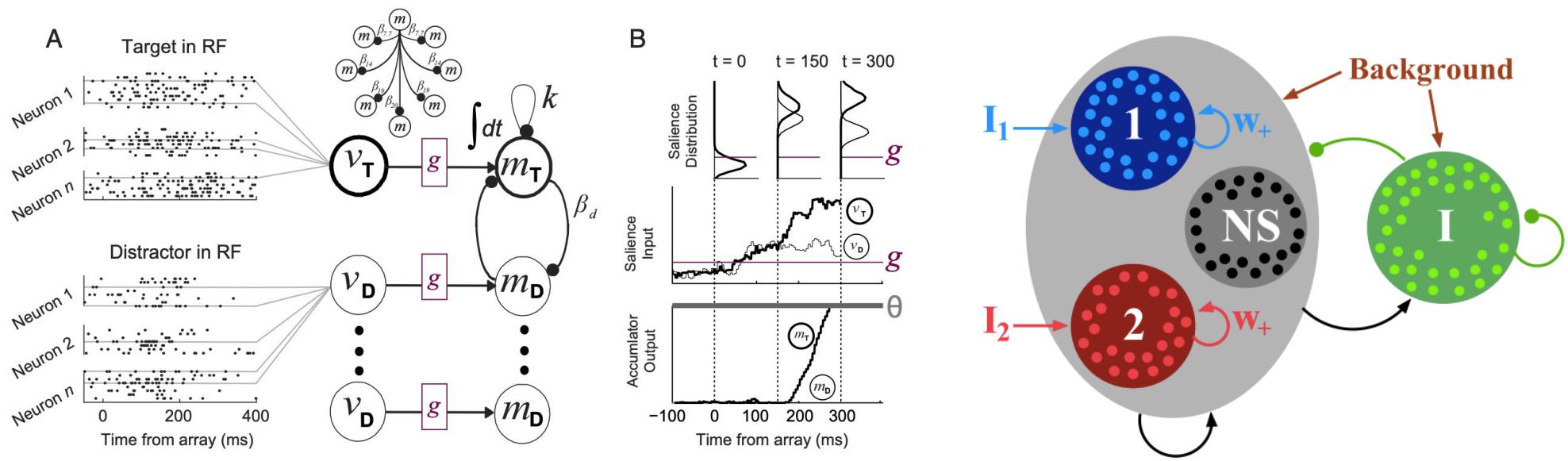
- monkeys (and humans) sometimes are correct, sometimes make errors (given the same stimulus array)
- monkeys (and humans) vary in their response times, sometimes slow, sometimes fast (given the same stimulus array)
- our models (computational and mathematical instantiations of theories) need to account 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

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

T. Bayes.

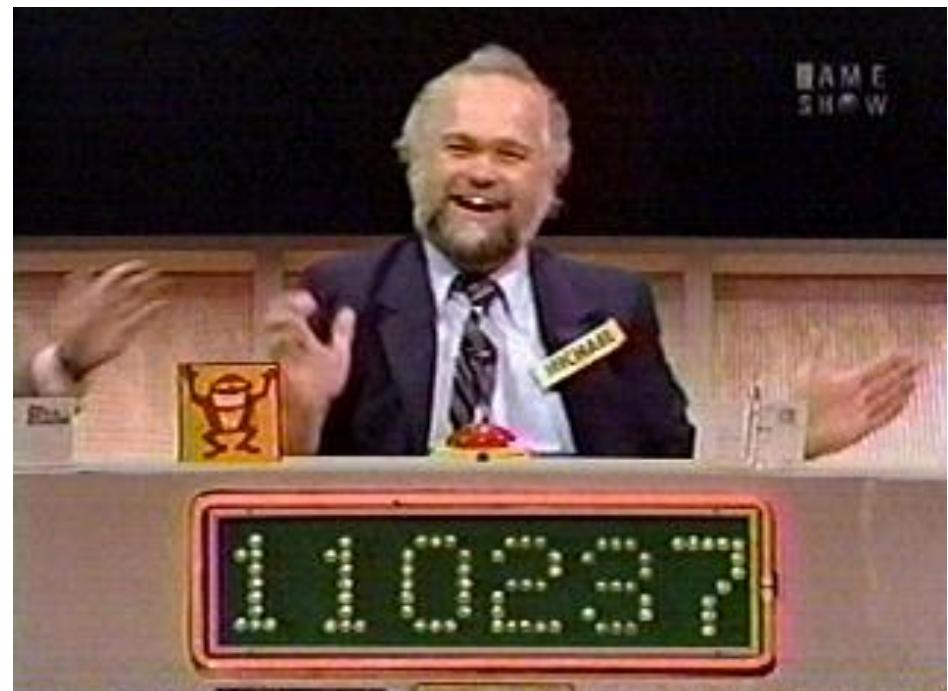
- we might use Bayes methods for parameter estimation (model fitting) and model comparison
- Bayesian methods make probabilistic statements about parameters and models, requiring the use of sophisticated random number generators to operate
- or we might use a genetic algorithm or simulated annealing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



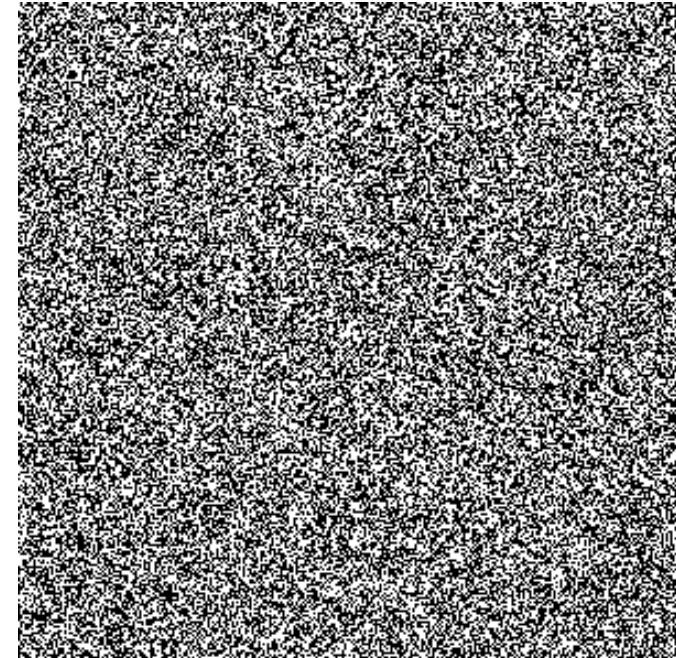
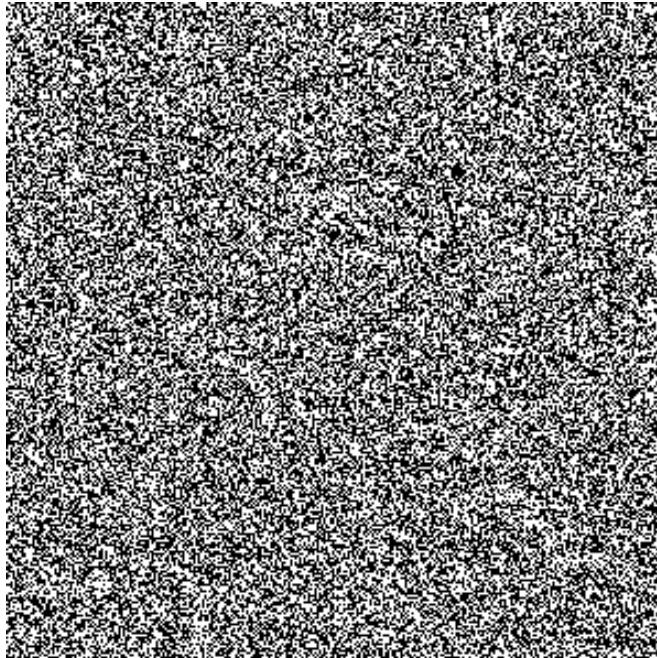
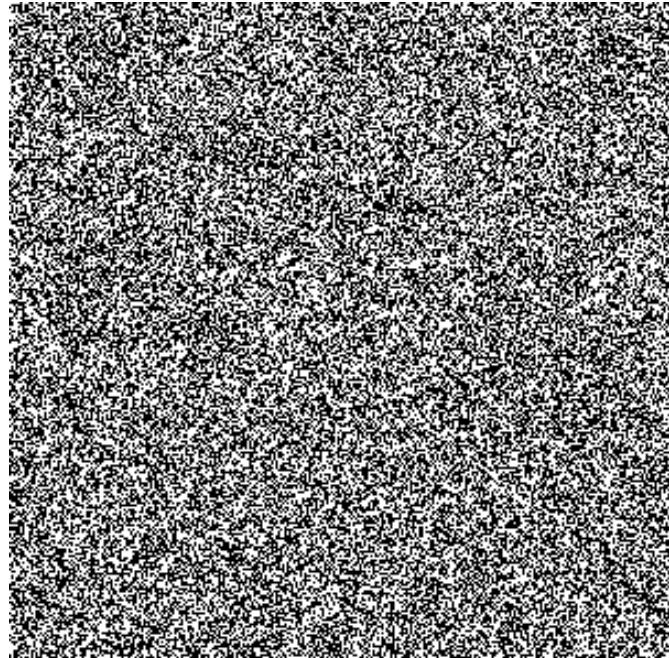
In 2007, the Tennessee lottery switched from drawing balls to a computer program for their pick-3 and pick-4 daily lottery.

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 ...”*

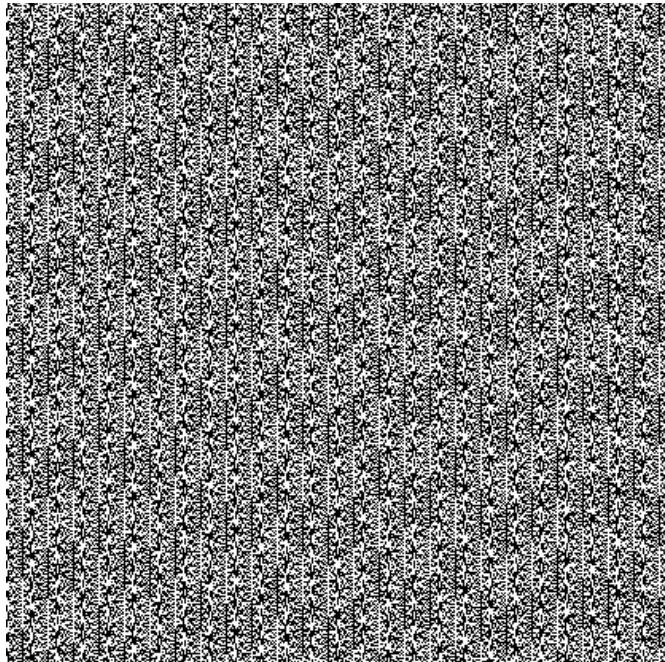
Imagine generating a random bit map image of 0's and 1's ...

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



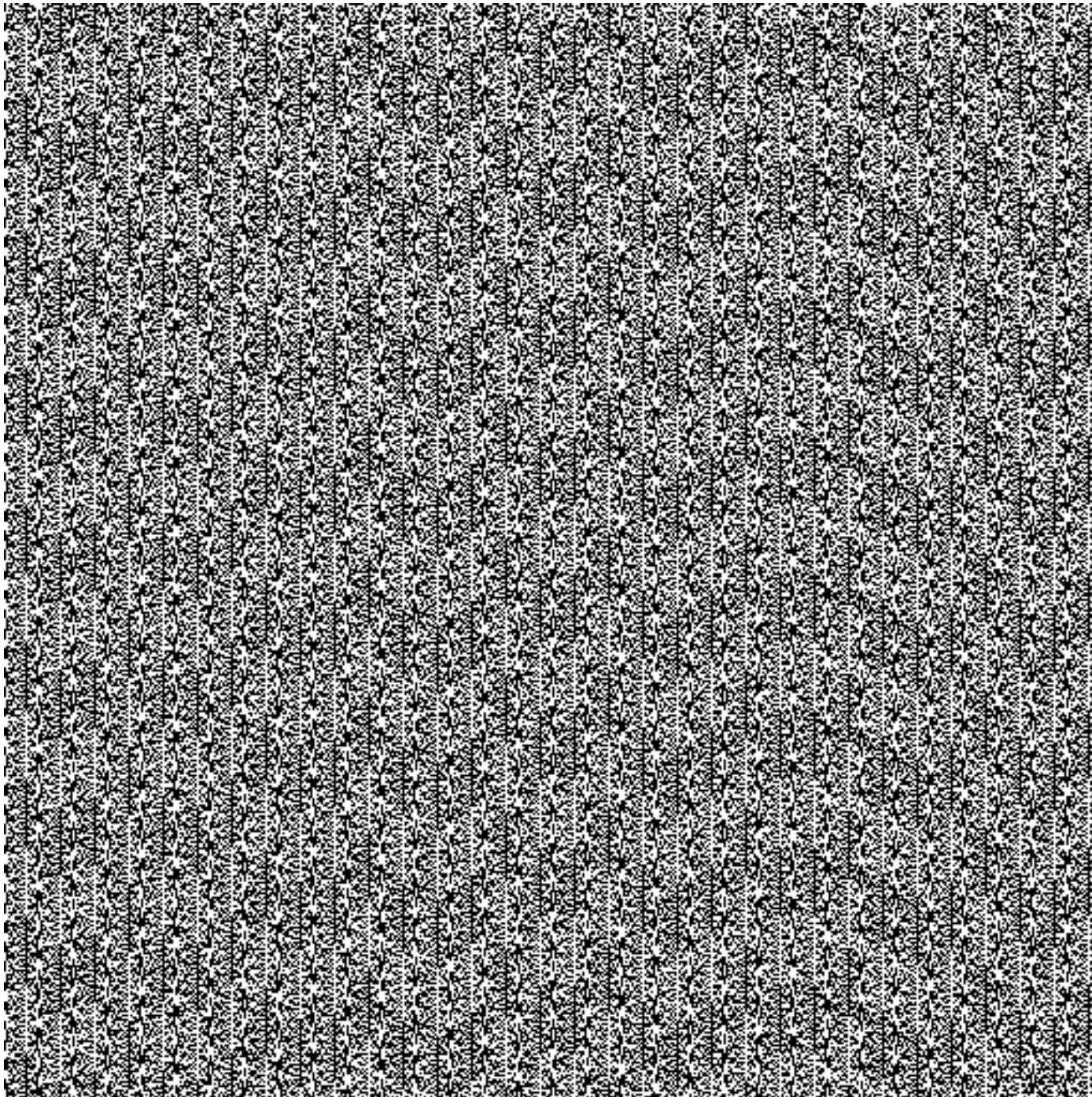
they should look something like this ...

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

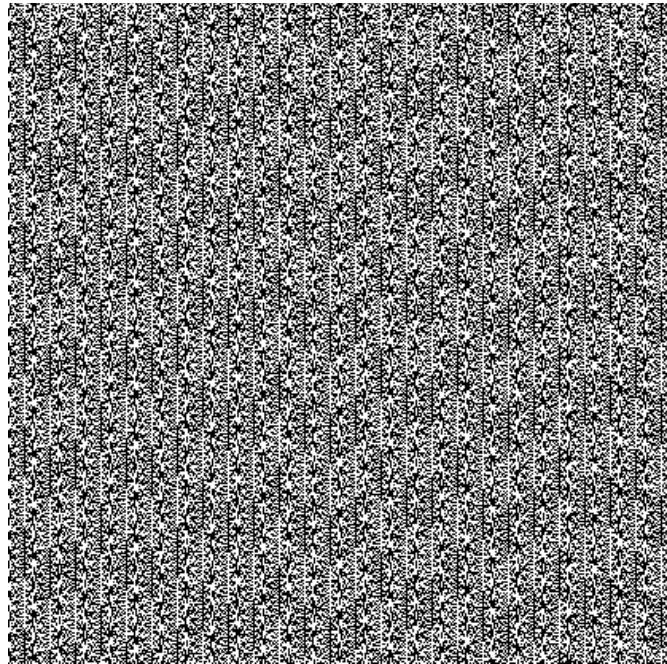


this clearly doesn't look random ...

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 ...

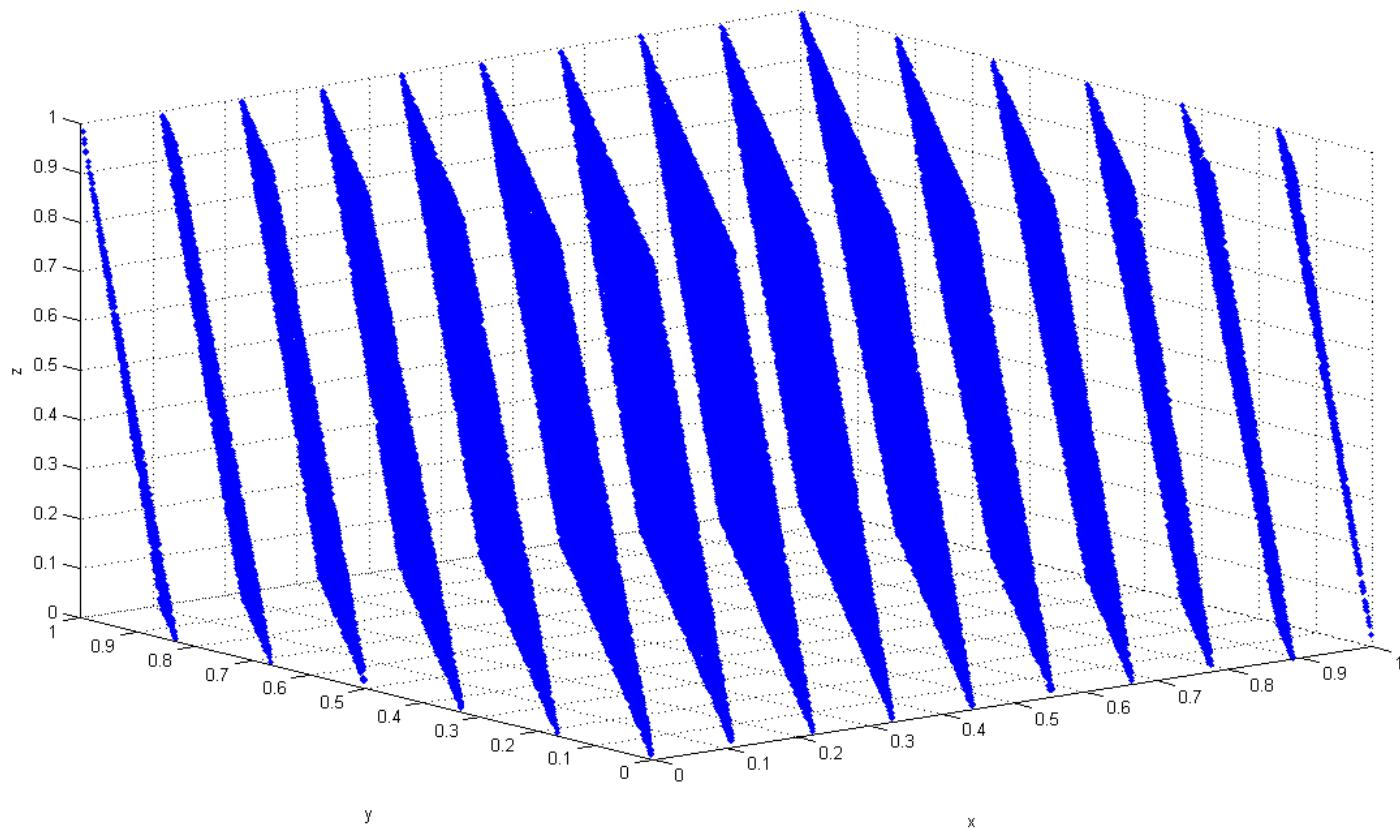
**IBM**

**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

The screenshot shows the Amazon product page for the book "A Million Random Digits with 100,000 Normal Deviates". The page includes the book cover, purchase options (Hardcover, Paperback, Other Sellers), Prime shipping information, and a "More Buying Choices" section.

**A Million Random Digits with 100,000 Normal Deviates** 0th Edition

by The RAND Corporation (Author)

★★★★★ 623 customer reviews

[Look inside](#)

**Paperback** \$44.00 - \$54.89

Hardcover    Other Sellers from \$44.00

Buy used    \$44.00

Buy new    **\$54.89**

Prime List Price: \$68.00 Save: \$13.11 (19%)  
11 New from \$54.89

Qty: 1

Add to Cart

Want it tomorrow, Oct. 9? Order within 1 hr 6 mins and choose One-Day Shipping at checkout. Details

Turn on 1-Click ordering

Ship to: Thomas J. Palmeri- Nashville

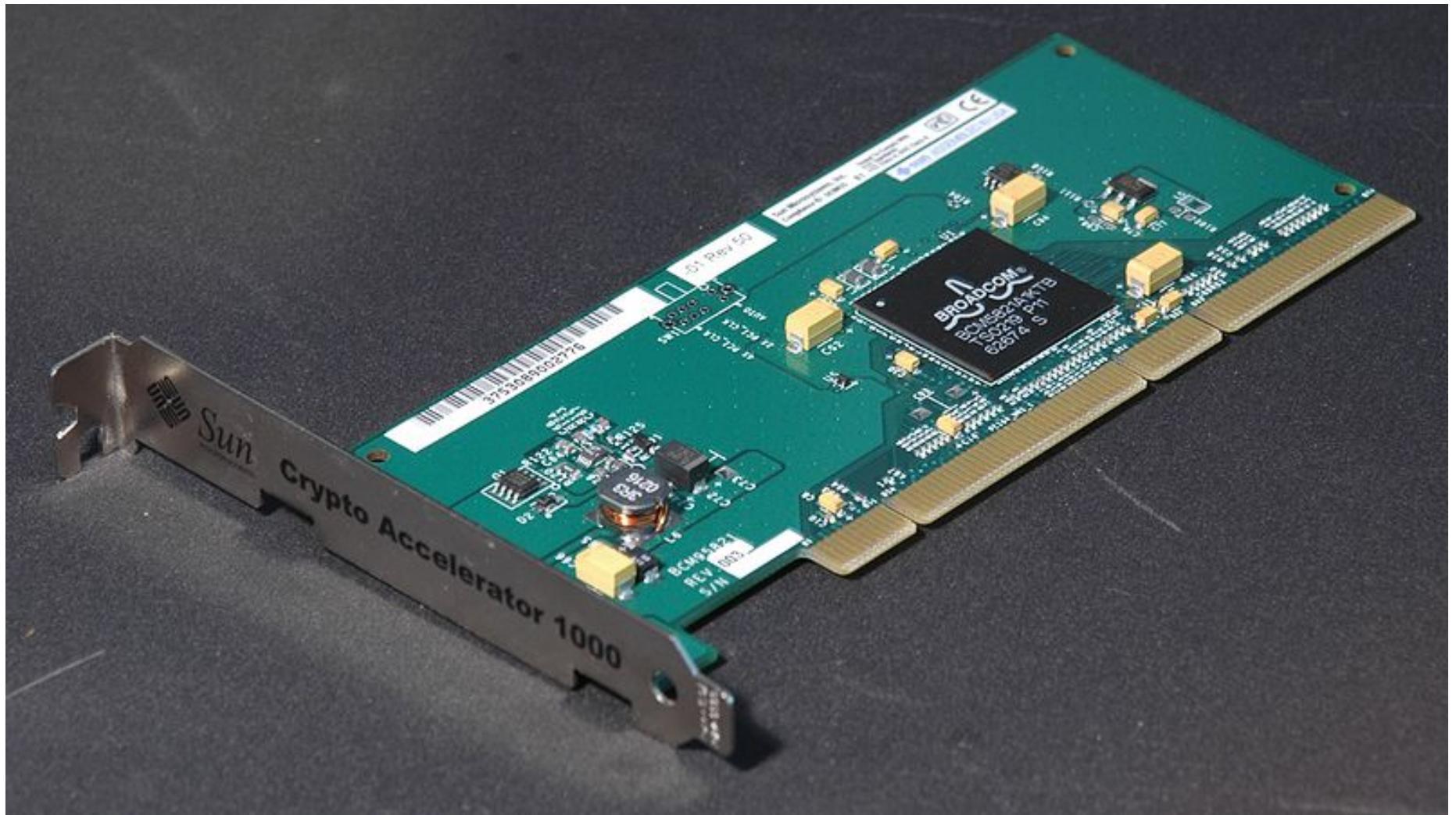
ISBN-13: 978-0833030474  
ISBN-10: 0833030477

More Buying Choices 25 used & new from \$44.00

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

[Home](#)   [Games](#)   [Numbers](#)   [Lists & More](#)   [Drawings](#)   [Web Tools](#)   [Statistics](#)   [Testimonials](#)   [Learn More](#)   [Login](#)

# RANDOM.ORG

Search RANDOM.ORG



[Search](#)

**True Random Number Service**

[www.random.org](http://www.random.org)

*based on atmospheric noise levels*

Inactive Plug-in

**HotBits: Genuine random numbers, generated by radioactive decay**

[www.fourmilab.ch/hotbits/](http://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

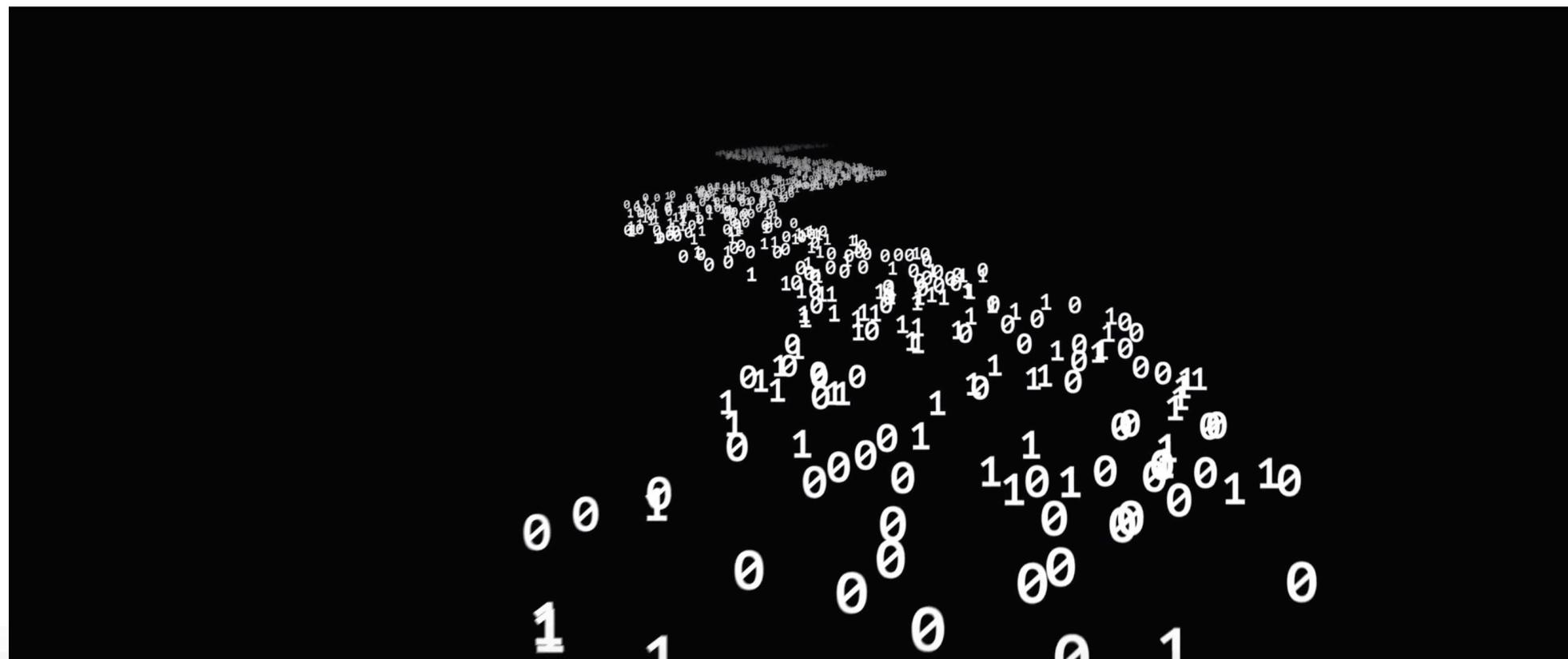


ANIL ANANTHAWAMY

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...10101111110101...010101110100...00110011111 ...



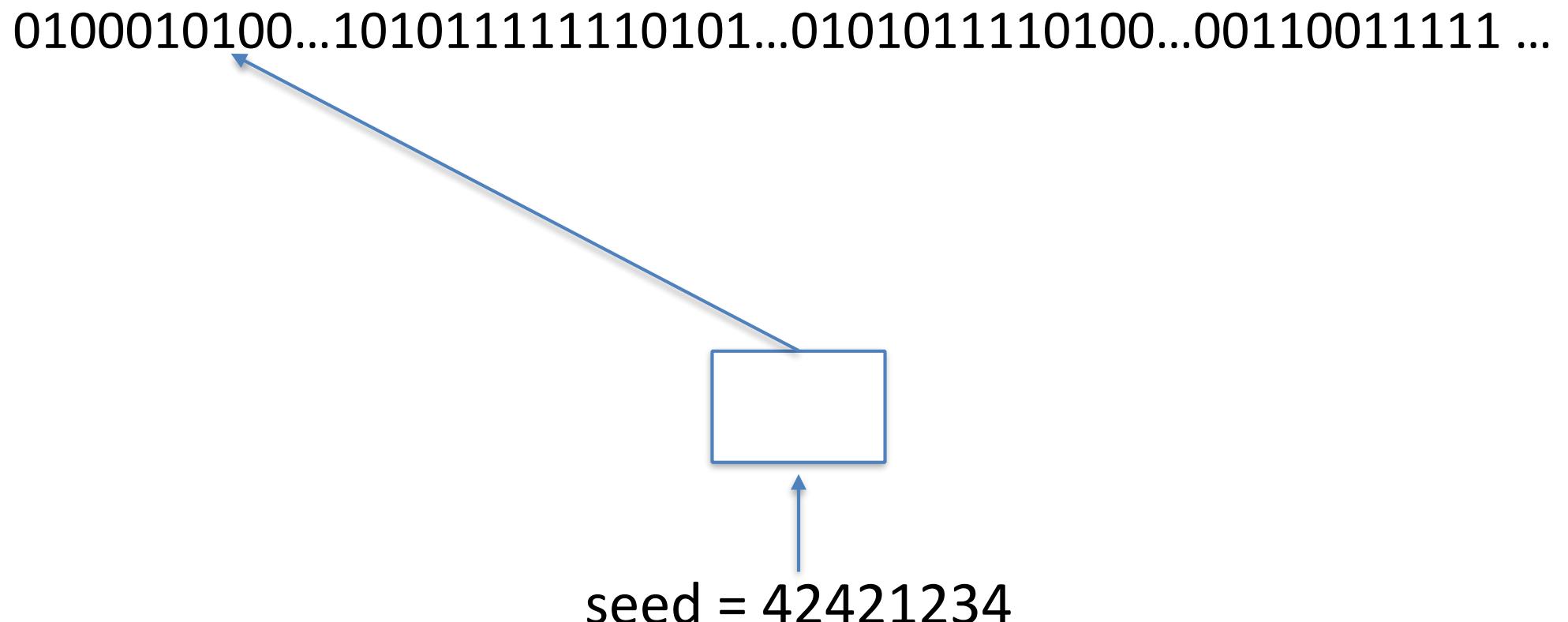
*if I know I'm here in the sequence,  
the rest of the sequence is perfectly predictable*

(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*



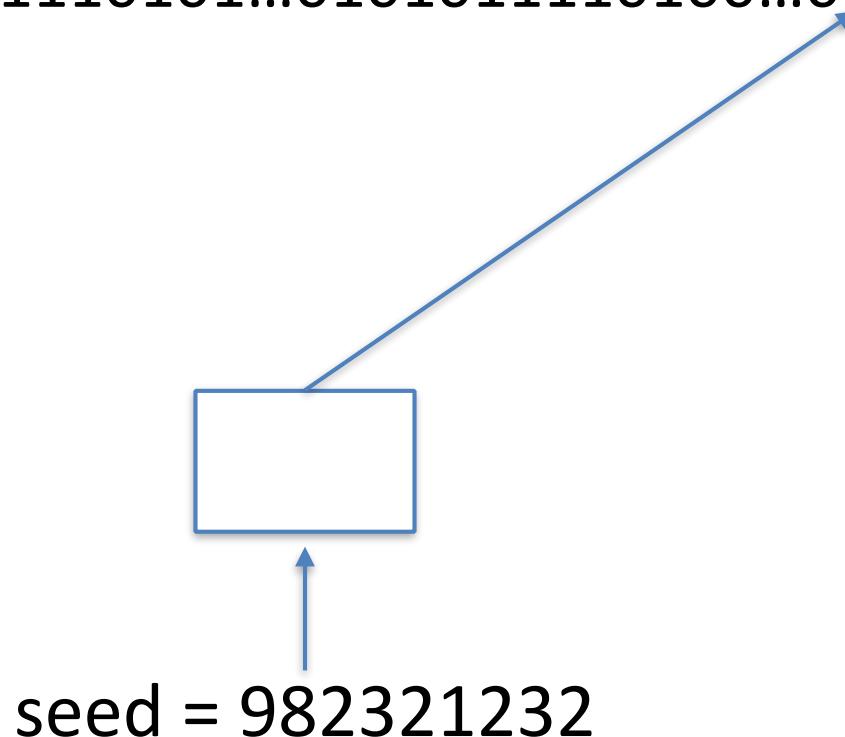
(without connection to a random physical process)

# 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...10101111110101...010101110100...0011001111 ...



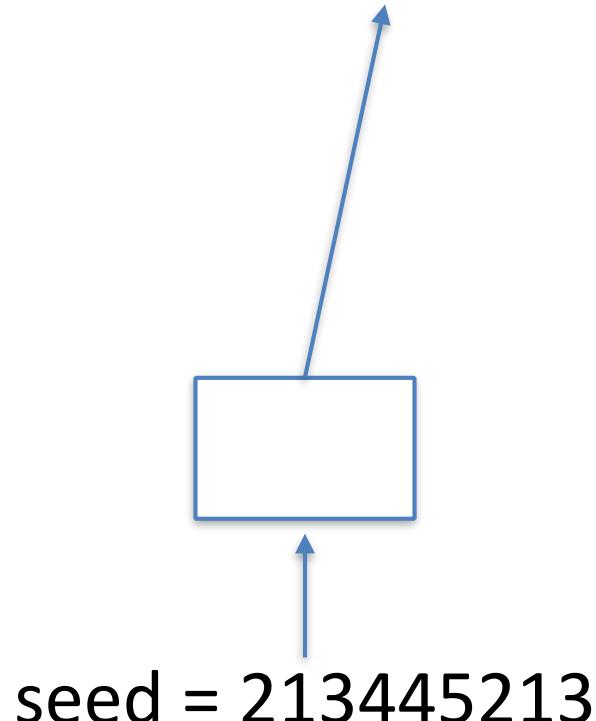
(without connection to a random physical process)

# 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...10101111110101...0101011110100...00110011111 ...



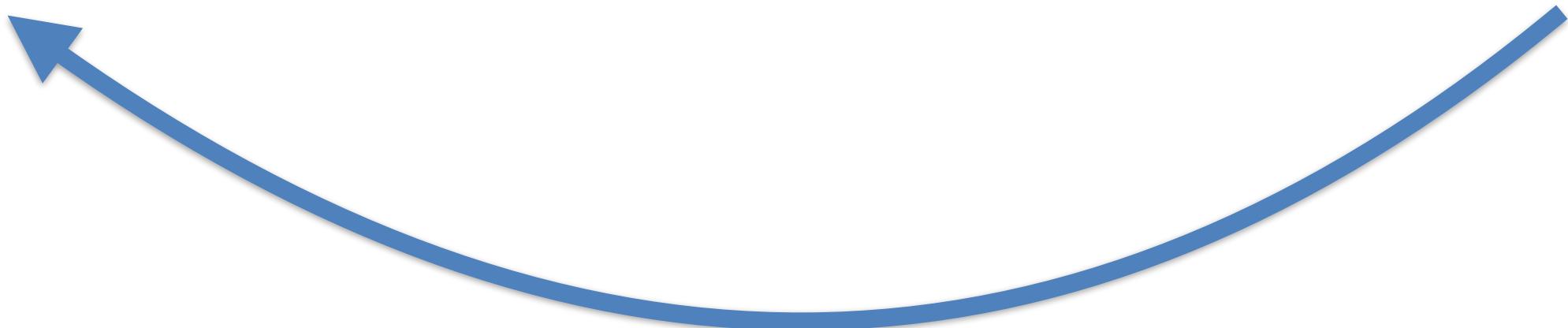
(without connection to a random physical process)

# random numbers on a computer

## pseudo-random number generators

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

0100010100...10101111110101...010101110100...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...10101111110101...010101110100...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...10101111110101...010101110100...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...10101111110101...010101110100...00110011111 ...

pseudo-random number generator   OR   real random process

1001111001...01111110100011...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...010101110100...0011001111 ...  
pseudo-random number generator   OR   real random process

1001111001...01111110100011...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

National Institute of Standards and Technology  
Information Technology Laboratory

SEARCH CSRC:  GO

ABOUT MISSION CONTACT STAFF SITE MAP

Computer Security Division CSD

Computer Security Resource Center CSRC

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  $\frac{1}{2}$ , 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 0001101111101001100000010110001111101  
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 if 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](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))
```

tuple

```
# sample from normal (mean 0, stdev 1)
```

```
c = R.normal()
```

tuple

```
d = R.normal(size=(6,2))
```

# 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  
0.8091308303813897  
[(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

**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
```

returns nanoseconds  
since January 1970

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

this is the max  
seed value

```
R.seed(seed)
```

returns  
remainder

# 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