Which of the following commands could give rise to this plot?

A  \( x = \text{np.random.randint}(0,10,\text{size}=(1000,)) \)
   \( \text{plt.hist}(x,\text{bins}=20) \)

B  \( x = \text{np.random.normal}(\text{size}=(1000,)) \)
   \( \text{plt.hist}(x,\text{bins}=20) \)

C  \( x = \text{np.random.randint}(0,10,\text{size}=(1000,)) \)
   \( \text{plt.plot}(x,\text{'rx'}) \)

D  \( x = \text{np.random.uniform}(\text{size}=(1000,)) \)
   \( \text{plt.hist}(x,\text{bins}=20) \)
Which of the following commands could give rise to this plot?

A  \( x = \text{np.random.randint}(0,10,\text{size}=(1000,)) \)
   \( \text{plt.hist}(x,\text{bins}=20) \)

B  \( x = \text{np.random.normal}(\text{size}=(1000,)) \)
   \( \text{plt.hist}(x,\text{bins}=20) \)

C  \( x = \text{np.random.randint}(0,10,\text{size}=(1000,)) \)
   \( \text{plt.plot}(x,'rx') \)

D  \( \star \)
   \( x = \text{np.random.uniform}(\text{size}=(1000,)) \)
This program should simulate Yahtzee, a dice game which requires five dice rolls. Which line should replace the `???`?

```
import numpy as np

``` ???

```
score( roll )

A roll = np.random.uniform( 5 )
B roll = np.random.choice( range( 5 ) )
C roll = np.random.randint( 1,7,size=(5,) )
D roll = np.random.randint( 5 )
```
This program should simulate Yahtzee, a dice game which requires five dice rolls. Which line should replace the ????

```python
import numpy as np
roll = np.random.randint( 1,7,size=(5,) )
score( roll )

A roll = np.random.uniform( 5 )
B roll = np.random.choice( range( 5 ) )
C roll = np.random.randint( 1,7,size=(5,) )
D roll = np.random.randint( 5 )
```
When Things Go Wrong
Common exceptions

- SyntaxError
- NameError
- TypeError
- ValueError
- ZeroDivisionError
- FileNotFoundError
- IndexError
- KeyError
- IndentationError
- Exception
Types of Bugs

- A few working definitions:
  - **Exceptions**—unusual behavior (although not necessarily unexpected behavior, particularly in Python)
A few working definitions:

- **Exceptions**—unusual behavior (although not necessarily unexpected behavior, particularly in Python)
- **Errors**—exceptions which cause the program to be unrunnable (cannot be handled at run time)
Types of Bugs

A few working definitions:

- **Exceptions**—unusual behavior (although not necessarily unexpected behavior, particularly in Python)
- **Errors**—exceptions which cause the program to be unrunnable (cannot be handled at run time)
- **Bugs**—errors and exceptions, but also miswritten, ambiguous, or incorrect code which in fact runs but does not advertise its miscreancy
# calculate squares
d = list(range(10))
while i < 10:
    d[i] = d[i] ** 2.0
    i += 1

Which error would this code produce?

A SyntaxError  
B IndexError  
C ValueError  
D NameError
# calculate squares
d = list(range(10))
while i < 10:
    d[i] = d[i] ** 2.0
    i += 1

Which error would this code produce?

A SyntaxError
B IndexError
C ValueError
D NameError★
Which of the following would produce TypeError?

A  '2' + 2
B  2 / 0
C  2e8 + (1+0j)
D  '2' * 2
Which of the following would produce `TypeError`?

A  '2' + 2 ★
B  2 / 0
C  2e8 + (1+0j)
D  '2' * 2
Program stack

- **Traceback**—listing of function calls on the stack at the time the exception arises
Program stack

- **Traceback**—listing of function calls on the stack at the time the exception arises

```python
def fun1():
    fun2()

def fun2():
    fun3()

def fun3():
    assert 1 == 2

fun1()
```
Traceback—listing of function calls on the stack at the time the exception arises
When Things Go Wrong

Traceback — listing of function calls on the stack at the time the exception arises

AssertionError

---

<ipython-input-1-b0cb5ad6fd6e> in fun1()
   1 def fun1():
   2     fun2()
   3     Assert 1 == 2

<ipython-input-1-b0cb5ad6fd6e> in fun2()
   4 def fun2():
   5     fun3()

<ipython-input-1-b0cb5ad6fd6e> in fun3()
   7 def fun3():
   8     assert 1 == 2
Handling Exceptions
Most of the time, we want errors to happen—but we may not want our program to crash (stop executing)!

```python
# calculate square roots
d = list(range(10))
r = []
for i in d:
    try:
        r[i] = sqrt(d[i])
    except:
        print('An error occurred.')
        break
```
Exception handling

- Most of the time, we want errors to happen—but we may not want our program to crash (stop executing)!
- We can tell Python to try a block of code, and it will run normally except if something goes wrong.

```python
# calculate square roots
d = list(range(10))
r = []
for i in d:
    try:
        r[i] = sqrt(d[i])
    except:
        print('An error occurred.')
        break
```
Most of the time, we want errors to happen—but we may not want our program to crash (stop executing)!

We can tell Python to try a block of code, and it will run normally except if something goes wrong.

```python
# calculate square roots
d = list(range(10))
r = []
for i in d:
    try:
        r[i] = sqrt(d[i])
    except:
        print('An error occurred.')
        break
```
Exception handling

- The advantage: you can handle the error and execution can proceed normally.
- The disadvantage: the traceback doesn’t appear automatically.

```python
# d = list(range(10))
i = 0
while i < len(d) + 1:
    try:
        d[i] = d[i] ** 2.0
        i += 1
    except:
        print('An error occurred.')```
Exception handling

- The advantage: you can handle the error and execution can proceed normally.
- The disadvantage: the traceback doesn’t appear automatically.
- This also doesn’t guard against errors or bugs which don’t raise an exception:

```python
d = list( range( 10 ) )
i = 0
while i < len( d )+1:
    try:
        d[ i ] = d[ i ] ** 2.0
        i += 1
    except:
        print( 'An error occurred.' )
```
try:
    x = 1 / 0
except:
    print("Division by zero occurred.")
denom = 0
while True:
    try:
        # Read int from console.
        denom = input()

        # Use as denominator.
        i = 1 / float(denom)
    except:
        print("non-numeric value entered")
    else:
        print(i)
    finally:
        if denom == 'q': break
Examples

```python
try:
    # the main code
except:
    # an error occurs
else:
    # but if no error occurs
finally:
    # in any case, this happens
```
If we lose the information on what went wrong, our response may not be appropriate.
If we lose the information on what went wrong, our response may not be appropriate.

What could have gone wrong in the code below?

```
try:
    filename = 'spring.data'
    datafile = open(filename, 'r')
    data = datafile.readlines()
except:
    print('Something went wrong.')
```
Examples

Use try at the finest degree of precision you can:

```python
filename = 'spring.data'
try:
    datafile = open( filename,'r' )
except:
    print( 'Unable to open file "%s".'%filename )
```

is better than

```python
filename = 'spring.data'
try:
    datafile = open( filename,'r' )
    for line in data:
        ...
except:
    ...
```
Numerical Error
\[
2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 + 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} + 2^{-5} + 2^{-6} + 2^{-7} + 2^{-8}
\]

\[
= 128 + 4 + \frac{1}{2} + \frac{1}{32} + \frac{1}{64} + \frac{1}{256}
\]

\[
= 132.55078125
\]
1.1_{10} 0001100110011001100110011001100110011001100110011010
0.8_{10} 1001100110011001100110011001100110011001100110011001
1.1−0.8_{10} 0011001100110011001100110011001100110011001100110011
1.1
0.8
1.1–0.8
0.3

000110011001100110011001100110011001100110011010
100110011001100110011001100110011001100110011010
001100110011001100110011001100110011001100111
0011001100110011001100110011001100110011001110100
0011001100110011001100110011001100110011001110100
1.1_{10} \quad 001100110011001100110011001100110011001100110011010
0.8_{10} \quad 1001100110011001100110011001100110011001100110011010
1.1-0.8_{10} \quad 001100110011001100110011001100110011001100110011010 100
0.3_{10} \quad 001100110011001100110011001100110011001100110011010 011
\Delta_{10} \quad \underbrace{000000000000}_{001}
Don’t compare directly:

- `a == b`  # never do this for floats!
- `np.isclose(a, b, rtol=1e-05, atol=1e-08)`
- `np.allclose(a, b, rtol=1e-05, atol=1e-08)`

Parameters:

- `rtol`  # relative tolerance (w/i percent)
- `atol`  # absolute tolerance
The number $0.1_{10}$ is written in binary as
The number $0.1_{10}$ is written in binary as

$$0.00011001100110011001100110011001100\ldots_2.$$
The number $0.1_{10}$ is written in binary as

$$0.0001100110011001100110011001100\ldots_2,$$

which the machine represents as

$$0.000110011001100110011001100\ldots_2.$$
The number $0.1_{10}$ is written in binary as

$$0.0001100110011001100110011001100_2,$$

which the machine represents as

$$0.00011001100110011001100_2.$$

The difference of these numbers is

$$0.0000000000000000000000000011001100_2.$$
The number $0.1_{10}$ is written in binary as
\[0.000110011001100110011001100\ldots_2,\]
which the machine represents as
\[0.00011001100110011001100_2.\]
The difference of these numbers is
\[0.000000000000000000000000000000000011001100\ldots_2,\]
rendered in decimal as about $0.000\ 000\ 095_{10}$. 
The number $0.1_{10}$ is written in binary as

$$0.0001100110011001100110011001100 \ldots_2,$$

which the machine represents as

$$0.00011001100110011001100_2.$$

The difference of these numbers is

$$0.0000000000000000000000000000011001100 \ldots_2,$$

rendered in decimal as about $0.0000000095_{10}$.

$$100 \text{ hr} \times 60 \frac{\text{min}}{\text{hr}} \times 60 \frac{s}{\text{min}} \times (10 \times 0.000000095_{10}) = 0.34 \text{ s}$$
The number \(0.1_{10}\) is written in binary as
\[0.000110011001100110011001100\ldots_2\]
which the machine represents as
\[0.00011001100110011001100_2\].

The difference of these numbers is
\[0.0000000000000000000000011001100\ldots_2\]
rendered in decimal as about \(0.000\ 000\ 095_{10}\).

\[
100 \text{ hr} \times 60 \frac{\text{min}}{\text{hr}} \times 60 \frac{\text{s}}{\text{min}} \times (10 \times 0.000\ 000\ 000\ 095_{10}) = 0.34 \text{ s}
\]
Which of the following expressions is liable to experience problems with numerical error? Assume all variables are defined and have appropriate type.

A. \( ( a / 1 \text{e}5 < 0 ) \)
B. \( ( b \leq 1.0 ) \)
C. \( ( c ^{0.5} ) / 2 \)
D. \( ( d == 0.4 ) \)
16-BIT FLOATING-POINT NUMBERS

(-1)^0 \ sign \ of \ quantity

0 \ sign \ bit

2^{10000100_2} \ exponent \ counted \ from \ -127

010000100 \ exponent

1._01011101000000000000000000000000b \ significand \ without \ leading \ bit

= (-1)^0 \times 2^{10000100_2} \times 01011101000000000000000000000000b

= (-1)^0 \times 2^{132-127} \times 1.01011101b

= (-1)^0 \times 2^5 \times 1.36328125

= 43.625