Numerical Python

Errors & Exceptions
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 ⋆
\[ x = \text{np.random.uniform}(\text{size}=(1000,)) \]
This program should simulate Yahtzee, a dice game which requires five dice rolls. Which line should replace the ???

```python
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
A few working definitions:

- **Exceptions**—unusual behavior (although not necessarily unexpected behavior, particularly in Python)
When Things Go Wrong

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)
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

```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

```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.
AssertionError

Traceback (most recent call last)
<ipython-input-1-b0cb5ad6fd6e> in <module>()
 ---> 10 fun1()

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

<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)!
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.
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.
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.' )
```
Examples

```python
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
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.

```python
try:
    filename = 'spring.data'
    datafile = open(filename, 'r')
    data = datafile.readlines()
except:
    print('Something went wrong.')```
If we lose the information on what went wrong, our response may not be appropriate.

What could have gone wrong in the code below?

```python
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

filename = 'spring.data'
try:
    datafile = open( filename,'r' )
    for line in data:
        ...
except:
    ...
```
Numerical Error