User Input

- Rather than having to change the values of the variables inside a script for each test case it would be nice if the user could enter those values from the keyboard.

```
In [1]: colour=input('Enter a colour: ')
Enter a colour: blue
In [2]: print(colour)
blue
In [3]: print(type(colour))
<class 'str'>
```

- The function *input* is given a string (known as a **prompt**) to display and always returns a value of type 'str'.

- Suppose you want a numeric value.

```
In [1]: number = input('Enter a number: ')
Enter a number: 1234
In [2]: print(number)
'1234'
In [3]: print(type(number))
<class 'str'>
```

- We want an integer, convert the string to an integer.

```
In [4]: number = int(number)
In [5]: print(number)
1234
In [6]: print(type(number))
<class 'int'>
```

•We could use the ***eval*** function to covert the string returned by ***input*** to a numeric value.

```
In [1]: number = eval(input(
                             'Enter a number: '))
Enter a number: 12.34
In [2]: print(number)
12.34
In [3]: print(type(number))
<class 'float'>
In [4]: number = eval(input(
                             'Enter a number: '))
Enter a number: 1234
In [5]: print(number)
1234
In [6]: print(type(number))
```

-

`<class 'int'>`A program has no control over what the user types on the keyboard.

- If the program needs a number and the user enters a string the result will probably be wrong.

```
In [4]: number = eval(input('Enter a number: '))
Enter a number: 'hello'
In [5]: print(number)
hello
In [6]: print(type(number))
<class 'str'>
```

•

•Notice that the string 'hello' is not converted to a number.
 If ***number*** holds a string and we tried to use ***number*** in an arithmetic expression we might get an error or something unexpected might happen.

```
In [1]: print(number * 3)
hellohellohello

In [2]: print(number / 2)
TypeError: unsupported operand type(s) for /:
'str' and 'int'
```

•

•Before using a value entered by a user we must verify that it is of the correct type. An example of this will be presented later.

What happens if the user enters some text that is not a number and is not enclosed in quotes?

```
In [1]: number = eval(input('Enter a number: '))
Enter a number: hello
NameError: name 'hello' is not defined
```

-
- This error comes from trying to evaluate ***hello*** as a variable name, which in this case has not been defined.

•We need a way for a program to catch an error like this whenever a user enters data.•Suppose we would like to get an integer input from a user.

```
# integerInput1.py
def getInt():
    number = input('Enter an integer: ').strip()
    if number != '':
        number = eval(number, {}, {})
        if type(number) is int:
            print('%d is an integer.' % number)
        else: print(number, 'is not an
integer!') else: print('Missing input!')
getInt()
```

•Notice the pairs of braces, one pair of braces tells *eval* it does not have access to **global** variables and the other pair tells *eval* it does not have access to **local** variables.

```
In[1]: getInt()
Enter an integer: 12
12 is an integer.
In[2]: getInt()
Enter an integer: 12.24
12.24 is not an integer!
In[3]: getInt()
Enter an integer:
Missing input!
In[4]: getInt() Enter
an integer: True
True is not an integer!
In[5]: getInt() Enter an
integer: 'hello' hello is
not an integer!
  In[1]: getInt()
```

```
Enter an integer: hello

/Users/me/Download/integerInput1.py in getInt()
      4 number = input('Enter an integer:
        ').strip()
      5 if number != '':
----> 6     number = eval(number, {}, {})
      7     if type(number) is int:
      8     print('%d is an integer.' % number)

<string> in <module>()

NameError: name 'hello' is not defined
```

- The ***getInt*** function needs to be modified to handle this case.
- ***NameError*** is a type of **exception.**
- A program can handle exceptions using a **'try-except'** block.

```
# integerInput2.py
```

```
def getInt():
    number = input('Enter an integer: ').strip()
    if number != '':
        try:
            number = eval(number, {}, {})
        except:  # handle the error
            print('Invalid input!')
        else:    # no error, do this
            if type(number) is int:
                print('%d is an integer.' % number)
            else:
                print(number, 'is not an integer!')
    else: print('Missing
        input!')

getInt()
In [1]: getInt()
Enter an integer: hello
```

```
Invalid input!
In [2]: getInt()
Enter an integer: 1234
1234 is an integer.
In [3]: getInt()
Enter an integer: False
False is not an integer!
In [4]: getInt()
Enter an integer:
Missing input!
In [5]: getInt()
Enter an integer: 12.23
12.23 is not an integer!
In [6]: getInt() Enter
an integer: 'hi' hi is
not an integer!
```

•Repeatedly request an integer until the user enters an integer.

```
# integerInput3.py
def getInt(prompt):
    while True:
        number = input(prompt).strip()
        if number != '':
            try:
                number = eval(number, {},{})
            except:  # handle the error
                print('Invalid input!')
            else:    # no error, do this
                if type(number) is int:
                    break
                else:
                    print(number, 'is not an integer!')
        else: print('Missing
input!') return number
print(getInt('Enter an integer: '))
```

- Sample output from the program:

```
Enter an integer:
Missing input!

Enter an integer: hello
Invalid input!

Enter an integer: 'hello'
hello is not an integer!

Enter an integer: 12.34
12.34 is not an integer!

Enter an integer: True
True is not an integer!
```

```
Enter an integer: -17
-17
```

- How could we loop, inputting any integer, and exit from the loop when there are no more integers to read?
- In class demo of readLoop.py

```
def getInteger(prompt):
    while True:
        number = input(prompt).strip()
        if number != '':
            try: number = eval(number, {},
                {})
            except: print(number, 'is not
                valid!')
            else:
                if type(number) is int:
                    break
                else:
                    print(number, 'is not an integer!')
          else:
      break return
      number
def displayList(heading, theList):
    print('\n%s' % heading)
    for element in theList:
        print(element)
    print('There are %d elements in the list.' %
          len(theList))

def displayTerminationMessage():
    print('''
Programmed by Stew Dent.
Date: %s
End of processing.''' % ctime())
```

```
def main():
    numbers = []
    number = getInteger("Enter an integer: ")
    while number != '': numbers.append(number)
    number = getInteger("Enter an integer: ")
    displayList(
    'The elements in the list of numbers are:',
    numbers)
    displayTerminationMessage()

main()
```

- Sample output from this program:

```
Enter an integer: 23

Enter an integer: -1
Enter an integer: 7

Enter an integer: 3

Enter an integer:

The elements in the list of numbers are:
23
-1
7
3
There are 4 elements in the list.

Programmed by Stew Dent.
Date: Fri Jun 1 10:25:54
2018 End of processing.
```

Draw an X

•Consider a program that draws an X in an **n** by **n** grid. The value of **n** must be greater than 2 and must be an odd integer (3 or 5 or 7, etc.). The program must ensure the input **n** meets these criteria. A function will validate the input and another function will draw the X. The user should be able to draw as many X's as desired before terminating the program.

•As an X cannot be drawn unless the user enters a valid value for **n** (which we will refer to as **size**) we will examine the validation function named ***getValidInt***.

```
def getValidInt(prompt, minX):
        while True:
        number = input(prompt).strip()
        if number != '':
            try:
```

```
            number = eval(number, {}, {})
        except:  # handle the error
            print('Invalid input!')
        else:    # no error, do this
            if type(number) is int:
                if number % 2 == 1:
                    if number >= minX:
                        break
                    else:
                        print('%d is not greater than %d!' %
                              (number, minX-1))
                else: print('%d is not odd!' %
                    number)
            else:
                print(number, 'is not an integer!')
    else: print('Missing
        input!')
return number
```

- An X consists of two diagonal lines in an n by n grid.
- Suppose n = 3, the row and column numbers range from 0 to 2.

Row \ Column	0	1	2
0	*		*
1		*	
2	*		*

•For one of the diagonals containing asterisks the row and column numbers are equal. For the other diagonal containing asterisks the row number plus the column number is equal to n – 1, which in this case is 2.

```
def drawX(size):
    """ size of x
    Draw an X in size by size grid.
    size - the size of the grid
    """
    # indices - the valid row and column indicies
    # row       - the current column index
    # column    - the current column index

    indices = range(size)
    for row in indices:
        for column in indices:
            if row == column or row + column == size - 1:
                print('*', end='')
            else:
                print(' ', end='') print()  # for next
        print to start on a new line
```

•The program contains a function named ***main*** that calls the other functions

```
def main():
    """
    The main function. """
    # prompt    - the prompt to dispay to the user # minX
    - the minimum size of an X to draw # playAgain -
    prompt for playAgain question # drawAnX    - true if
    another X is to be drawn print('\n' + '-' * 80) prompt
    = 'Enter the size of the X (odd integer > 2): '
    playAgain = 'Draw another X? (Y/N): ' minX = 3 drawAnX
    = True while drawAnX:
        drawX(getValidInt(prompt, minX))
        drawAnX = getBoolean(playAgain)
    print("""
Programmed by Stew Dent.
Date: %s.
End of processing.""" % ctime())
```

•The function ***getBoolean*** repeatedly asks the user to enter a value until the user enters a word that begins with a Y or N.

```
def getBoolean(prompt):
    # prompt - the prompt to display to the user
    # data   - the input from the user
    # result - the return value: True if user enters
yes and
    #           False if user enters no
    while True:
        data = input(prompt).strip()
        data = data[0] if data == 'y'
        or data == 'Y':
            result = True
            break
        elif data == 'n' or data == 'N':
            result = False
            break
        else: print(data, 'is not a valid
            response!')
    return result
```

•The main program, not to be confused with the function named *main*, consists of two statements:

```
from time import ctime
main()
```

•Sample output from the draw an X program:

```
-----------------------------------------------

Enter the size of the X (odd integer > 3): 3
*   *
  *
*   *
Draw another X? (Y/N):     n

Programmed by Stew Dent.
Date: Fri Jun 1 15:34:29 2018.
End of processing.
```

•The following program inputs numbers until the user enters crtl-D, which raises an EOFError exception. Use this condition to terminate the loop. (eof.py)

```
from time import ctime smallest = 2**31-1       # compute
largest integer value largest = -2**31         # compute
```

```
smallest integer value count = 0 total = 0.
# force total to be a float loop = True  # continue
looping until loop becomes false
while loop:
    try:
        num = input('Enter a number: ')
    except EOFError:  # handle end of file error
        loop = False
    else:  # no error, do this
        try:
            num = eval(num, {}, {})
        except:  #  handle error
            print('Invalid input!')
        else:  # no error, do this
                      print ('%8.2f' % num)
                      total += num count +=
                      1
                       if num > largest:
                largest = num
                       if num < smallest:
                smallest = num
average = total / count
```

```
print(""" The
sum is %.2f
The average is %.2f
The smallest number is %.2f
The largest number is %.2f
""" % (total, average, smallest, largest))

print(""" Programmed by
Stew Dent.
Date: %s.
End of processing.""" % ctime())
```

Sample output:

```
Enter a number: 22
   22.00
Enter a number: 7
    7.00
Enter a number: 12
   12.00 Enter
a number:

The sum is 41.00
The average is 13.67
The smallest number is 7.00
The largest number is 22.00
```

```
Programmed by Stew Dent.
Date: Fri Jun 1 20:44:10 2018.
End of processing.
```

•This program can be rewritten to use a list and some python functions as follows (eof1.py):

```
from time import ctime

numbers = []
loop = True
while loop:
    try:
        num = input('Enter a number: ')
    except EOFError:  # handle end of file error
        loop = False
    else:  # no error, do this
        try:
            num = eval(num, {},{})
        except:  #  handle error
        print 'Invalid input!' else:
        # no error, do this
        print('%8.2f' % num)
        numbers.append(num)
```

```
total = sum(numbers)
average = float(total) / len(numbers)
print(""" The
sum is %.2f
The average is %.2f
The smallest number is %.2f
The largest number is %.2f
""" % (total, average, min(numbers), max(numbers)))

print("""
Programmed by Stew Dent.
Date: %s.
End of processing.""" % ctime())
```

- <u>NOTE</u>: Each of the functions ***sum***, ***min*** and ***max*** need a loop inside them, making the second version of the program less efficient than the first version.
- Sample output is shown on the next page.

```
Enter a number: 1
    1.00
```

```
Enter a number: 5
    5.00
Enter a number: -3
   -3.00
Enter a number: 99
   99.00
Enter a number: 2
    2.00
Enter a number: 7
    7.00
Enter a number: 4
    4.00
Enter a number:

The sum is 115.00
The average is 16.43
The smallest number is -3.00
The largest number is 99.00

Programmed by Stew Dent.
```

Date: Sat Jun 2 11:12:43 2018.Pyplot tutorial

- *matplotlib.pyplot* is a collection of command style functions that make matplotlib work with better quality

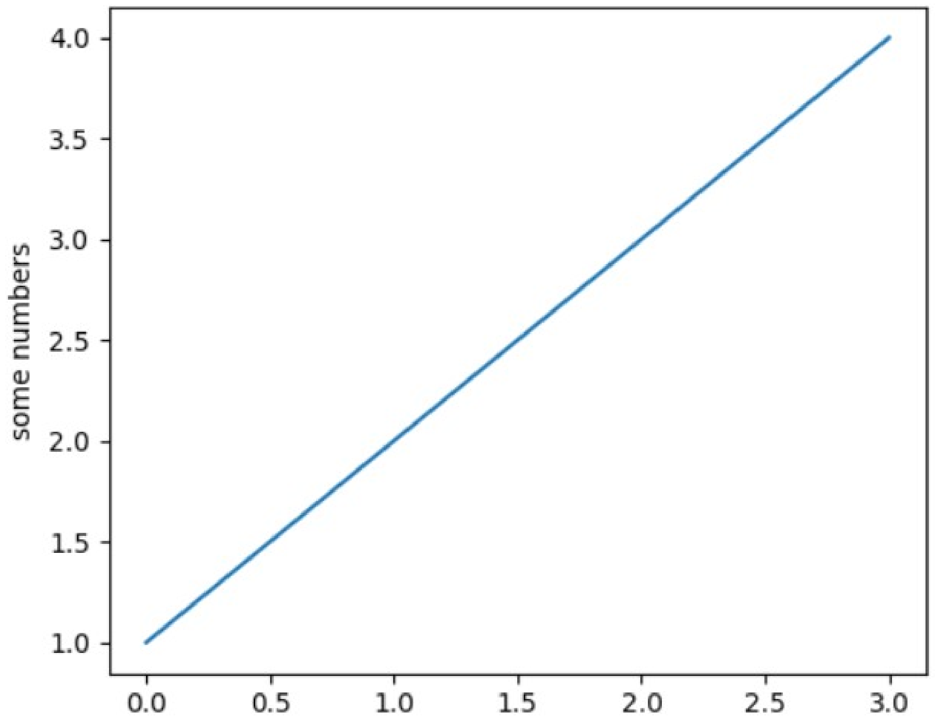
- Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

- In matplotlib.pyplot various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes

(please note that "axes" here and in most places in the documentation refers to the axes part of a figure and not the strict mathematical term for more than one axis).

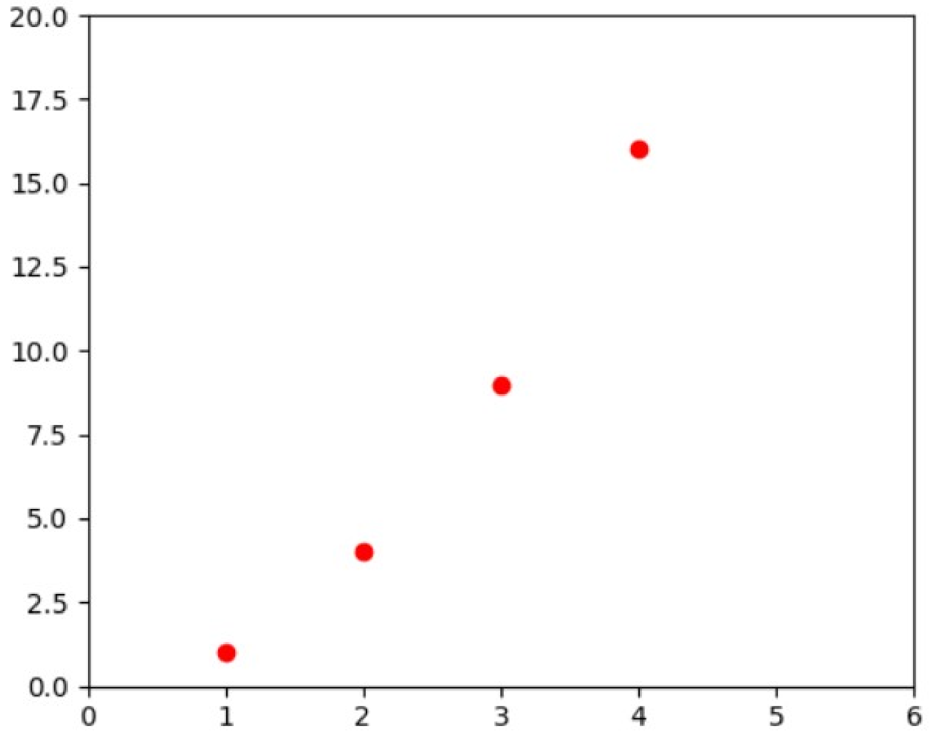
```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4])
plt.ylabel('some numbers')
plt.show()
```

(Source code, png, pdf)



```python
import matplotlib.pyplot as plt
plt.plot([1,2,3,4], [1,4,9,16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```

(Source code, png, pdf)

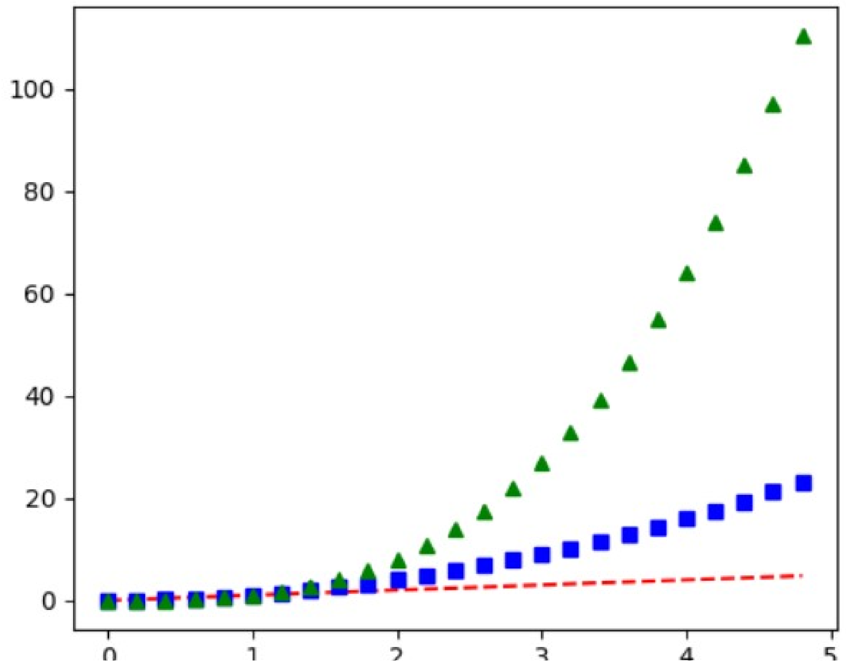


```
import numpy as np
import matplotlib.pyplot as plt

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```

(Source code, png, pdf)



100
80
60
40
20
0
0
1
2
3
4
5