Gather simple user input, while loops, try...catch statements, and introduction to importing modules

Input() function...

- Get user input from a prompt
- Can do some basic formatting...

In [ ]:
```
name = input('What is your name? ')
print(name)
```

Use of the 'sep' keyword in the print statement to determine what character goes in between phrases

In [ ]:
```
name = input('What is your name? ')
age = input('What is your age? ')
print(name, 'is', age, 'years old', sep=' ')
print(name, age, sep=':')
```

Dealing with numeric input (which is a string by default)....

- Best practice: don't just cast the str as an int because a user might input the wrong kind of data...or your data set might have a field that is miscoded (i.e. you encounter a name instead of a number)
- Better practice is to first test to see if a variable is a viable number, and if so, then deal with it...

note that input returns a string object, even if the input is a number

In [ ]:
```
type(age)
```

In [ ]:
```
# first a bad example...this will go wrong if the user enters a non-numeric value
age = int(input('How old are you?'))
type(age)
```

In [ ]:
```
# a more robust way to deal with input
age = input('How old are you? ')
if age.isdigit():
  print('You are', age, 'years old')
else:
  print('user input error!')
```

import statement to gain access to more functionality

- a module contains python classes or just functions
- a class defines the functionality of an object
- similar to #include header_file in C/C++ (if people are familiar with that)
- or setting a path to a toolbox in Matlab
- allows you to bring in a 'module' that supports specialized functionality (e.g. special math functions, etc)
- also allows for expanding the functionality of python - anyone can write a module to support new applications
- for the moment just a simple example...will build on this in the coming classes

In [ ]:
```
# simple example
import math
print(math.pi)
```

In [ ]:
```
# little more complex...cos method
print(math.cos(math.pi))
```

While loop syntax

- if you know exactly how many times you want to do something, then you can use a for loop
- if you don't know how many times you need to repeat something, then use while statement

Let's try some while loops, and we'll also import the "random" module, which has functionality to generate random numbers based on different distributions (uniform, gaussian, etc).

- random.random will generate random numbers from a uniform distribution over the interval [0,1]

In [ ]:
```
# import the entire random module
import random
```

In [ ]:
```
for i in range(0,6):
  # random.random will return a number between 0 and 1
  if random.random()<=.5:
    print('heads')
  else:
    print('tails')
```

In [ ]:
```
x = random.random()

# draw random numbers, print, and repeat until we get a draw >= .7
while x<.95:
  x = random.random()
  print(x)
```

Another example that will ask for user input until an appropriate value is provided - uses a boolean flag to keep looping until valid user input if provided...

In [ ]:
```
get_input = True

while get_input:
  age = input('What is your age? ')

  if age.isdigit():
    age = int(age)
    print('You will be', age + 10, 'in 10 years!')
    get_input = False

  else:
    print('User input error - please enter a valid integer')
```

```
What is your age? r3
User input error - please enter a valid integer
What is your age? rr
User input error - please enter a valid integer
What is your age? tt
User input error - please enter a valid integer
What is your age? 4t
User input error - please enter a valid integer
What is your age? 44
You will be 54 in 10 years!
```