REMINDER: help sessions

Mondays 3:50-5:00

this room (or WH 113)

Homework 1

posted on Brightspace

due next Wed (Sep 7)

only worth 5 points (other assignments will be longer and worth more points) - mainly to check that everyone has Python installed properly

download from Brightspace

```
Modules.ipynb
LogicalTypes.ipynb
StringsRegularExpressions.ipynb
ListsTuples.ipynb
SetsAndDictionaries.ipynb
```

a Python file
(that uses other modules)   collection of modules

Modules (and Packages)

`Modules.ipynb`

base Python has a small number of functions

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

https://docs.python.org/3/library/functions.html#built-in-funcs

most functions are in modules that are imported

the basic Python build comes with a number of modules and packages : https://docs.python.org/3/library/

HOMEWORK 2 will ask you to play around with some of these

e.g.,

`string` (common string operations)
`re` (regular expressions)
`math` (common math operations)
`cmath` (common math operations with complex numbers)
`datetime` (date and time)
`pprint` ("pretty" printing)
`fractions` (rational number operations)
`statistics` (basic stats operations)
`time` (time access and time conversions)
`turtle` (turtle graphics)
and many more ...

most functions are in modules that are imported

```
import math
```

https://docs.python.org/3/library/math.html

functions in `math` module: sqrt(), exp(), log(), log2(), log10(), combinations, permutations, trigonometric functions, special functions (erf, gamma), constants (pi, e, infinity, nan)

ways to import a module

```
import math
```

need to reference the module name

```
print(math.sin(math.pi/2))
```

function in
math module

constant in
math module

ways to import a module

alias (helps code readability)

```
import math as m
```

can be anything, but there are convention
e.g., `m` = math, `np` = numpy

```
print(m.sin(m.pi/2))
```

ways to import a module

```
from math import sin, pi
```

if you are just using a few function, constants

```
print(sin(pi/2))
```

ways to import a module

```
from math import *
```

imports everything (NOT RECOMMENDED)

```
print(sin(pi/2) + cos(pi/4)
      + exp(1) + log10(100))
```

ways to import a module

```
import math
import math as m
from math import sin
from math import *
```

Best Practices

NOT this

we will use a number of Python modules (and packages) course

base Python includes a number of modules
https://docs.python.org/3/library/index.html

Homework 2 **(Due Wed Sep 14th)**
explore one of these base Python modules
(`datetime`, `pprint`, `fractions`, `statistics`, `time`, or `turtle`)

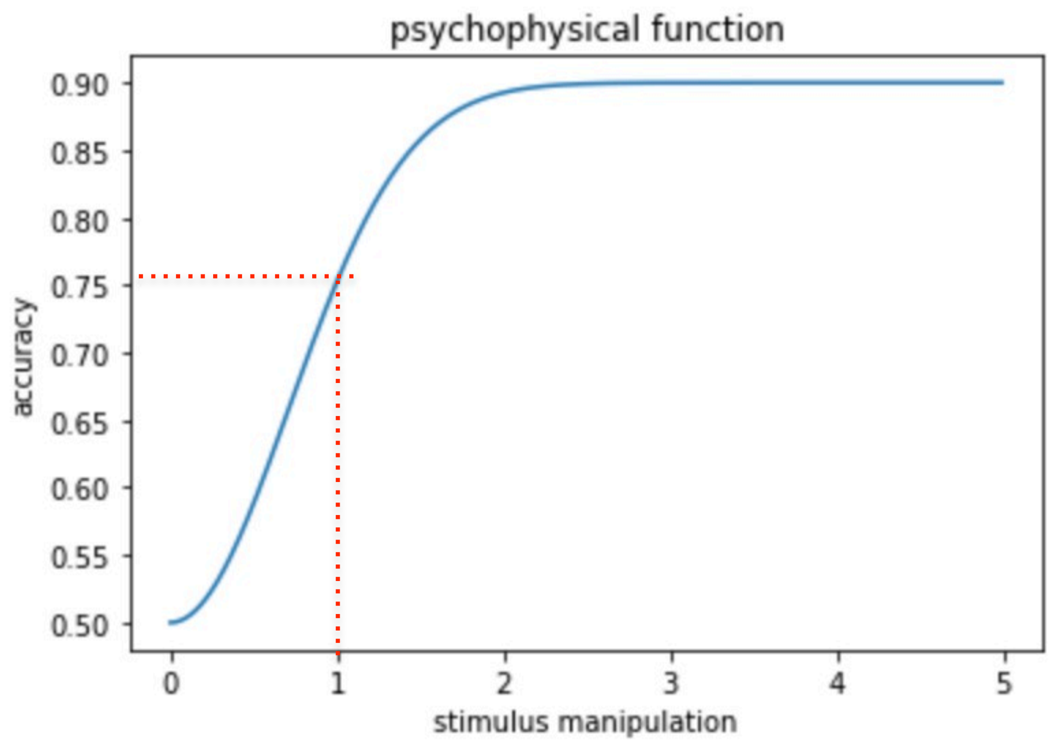
an important part of scientific computing is translating mathematical expressions (processing, analyses, modeling, visualization, and more) into Python code - here's an example:

$$\psi(x) = \gamma + (1 - \gamma - \lambda)F(x)$$

$$F(x) = 1 - \exp\left[-\left(\frac{x}{\alpha}\right)^{\beta}\right]$$

Note: $\exp(x)$ is the same as to e^x

`exp` is in the `math` module



here, we're just calculating values; later, we'll write these as functions

$$\psi(x) = \gamma + (1 - \gamma - \lambda)F(x)$$

$$F(x) = 1 - \exp\left[-\left(\frac{x}{\alpha}\right)^{\beta}\right]$$

$\alpha, \beta, \gamma, \lambda$ are parameters - assume the following values:

$\alpha = 1$

$\beta = 2$

$\gamma = .5$

$\lambda = .1$ note that `lambda` is a reserved word in Python

assume x = 1, calculate (and print) the value of $\psi(x)$

here, we're calculating <u>one</u> value; later we'll calculate an <u>array</u> of values

α, β, γ, λ are parameters - assume the following values:

$\alpha = 1$

$\beta = 2$

$\gamma = .5$

$\lambda = .1$

assume x = 1, calculate (and print) the value of $\psi(x)$

```
alpha = 1.
beta = 2.
gam = .5
lam = .1


x = 1.
```

need to create the variables before the code implementing the equations

$$\psi(x) = \gamma + (1 - \gamma - \lambda)F(x)$$

$$F(x) = 1 - \exp\left[-\left(\frac{x}{\alpha}\right)^{\beta}\right]$$

I've written the equations this way,

$$F(x) = 1 - \exp\left[-\left(\frac{x}{\alpha}\right)^{\beta}\right]$$

$$\psi(x) = \gamma + (1 - \gamma - \lambda)F(x)$$

I've written the equations this way,
but in code, we need to evaluate this way

$$F(x) = 1 - \exp\left[-\left(\frac{x}{\alpha}\right)^{\beta}\right]$$

I've written the equations this way,
but in code, we need to evaluate this way

```
import math as m

F = 1 - m.exp(-(x/alpha)**beta)
```

$$F(x) = 1 - \exp\left[-\left(\frac{x}{\alpha}\right)^{\beta}\right]$$

I've written the equations this way,
but in code, we need to evaluate this way

$$\psi(x) = \gamma + (1 - \gamma - \lambda)F(x)$$

```
import math as m

F = 1 - m.exp(-(x/alpha)**beta)

psi = gam + (1 - lam - gam)*F

print(psi)
```

```
import math as m

alpha = 1.
beta = 2.
gam = .5
lam = .1

x = 1.

F = 1 - m.exp(-(x/alpha)**beta)
psi = gam + (1 - lam - gam)*F

print(psi)
```

```
import math as m


x = 1.
```

NEVER hard code
ALWAYS use (meaningful) variables

```
F = 1 - m.exp(-(x/1.)**2.)
psi = .5 + (1 - .1 - .5)*F


print(psi)
```

Why?

- making code readable and understandable
- making code useable and changeable
- easier to debug
- required in this class (Jason will watch for this)

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2} \right)$$

if the equation has numbers in it (like the 2s in the equation for a Gaussian) then your code should have 2s in it as well

```
import math as m


mn = 0.

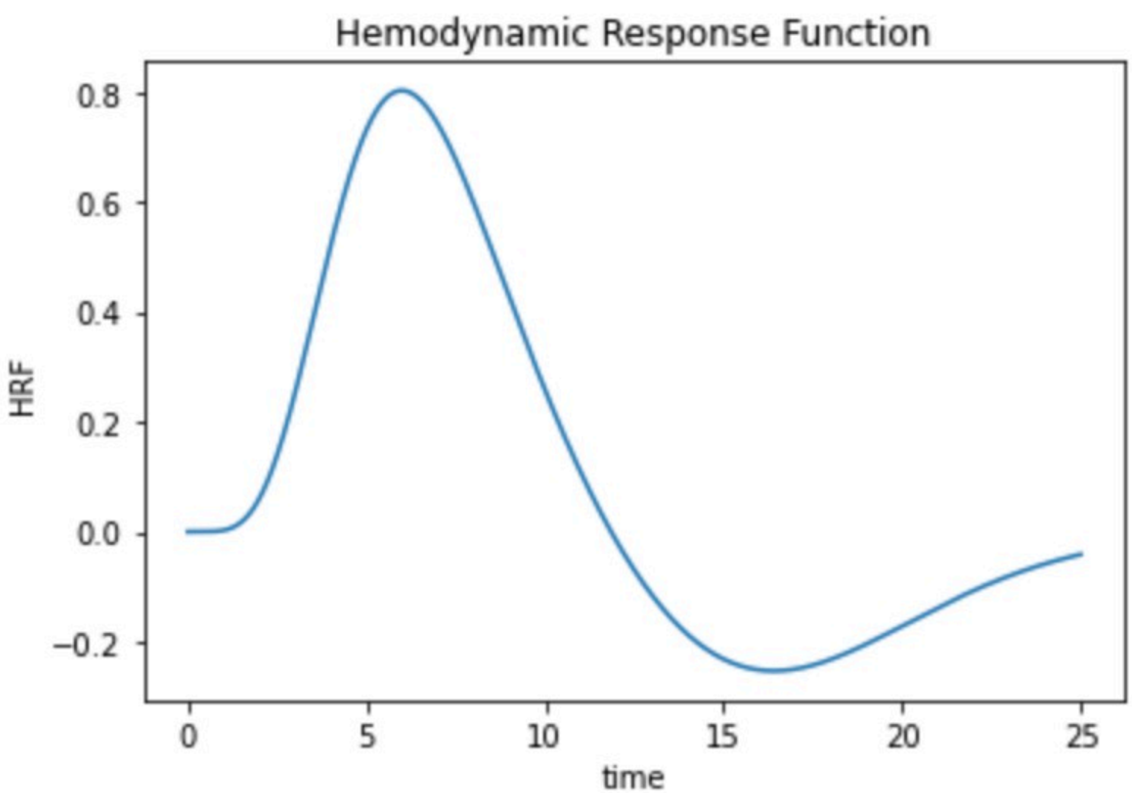
sd = 1.0

x = 0.

p = (1/m.sqrt(2*m.pi*(sd**2))) *
    m.exp(-((x-mn)**2)/(2*(sd**2)))
```

adapted from code in `JupyterNotebooks.ipynb`

Homework 2 (Due Wed Sep 14th)
asks you to implement in Python equation
for example hemodynamic response function
used in fMRI analyses (not graph it)



Boolean/Logical Types
Boolean/Logical Operators

`LogicalTypes.ipynb`

Logical/Boolean Types

```
True
False

x = (1 == 1)
type(x)

x = 2 + (1==1)
type(x)
```

can be useful for tallying the number of true instances

https://docs.python.org/3/library/stdtypes.html

Logical and Relational Operators

```
x = 2
y = 1
z = 3


x > y
y > z
```

Relational Operators

`>`	greater-than
`<`	less-than
`>=`	great-than-or-equal-to
`<=`	less-than-or-equal-to
`==`	equivalent (equal)
`!=`	not-equivalent (not equal)

Be careful using `==` and `!=` with real numbers

Does $\left(\sqrt{3}\right)^2 = 3$?

Be careful using `==` and `!=` with real numbers

Does $\left(\sqrt{3}\right)^2 = 3$?

```
(m.sqrt(3))^2 == 3
(m.sqrt(3))^2 – 3
```

What's going on?

How else might you check "equivalence"?

rough equivalence with reals

How else might you check “equivalence”?

```
epsilon = 0.00000001
abs((sqrt(3))^2 – 3) < epsilon
```

why is `abs()` needed?

equivalence within some tolerance (epsilon)

Logical Operators

```
x = 2
y = 1
z = 3
not (x > y)
```

Logical **Not**

A	not A
F	T
T	F

Logical Operators

```
x = 2
y = 1
z = 3
(x > y) and (y > z)
```

Logical **And**

A	B	A and B
F	F	?
F	T	?
T	F	?
T	T	?

Logical Operators

```
x = 2
y = 1
z = 3
(x > y) and (y > z)
```

Logical **And**

A	B	A and B
F	F	F
F	T	F
T	F	F
T	T	T

Logical Operators

```
x = 2
y = 1
z = 3
(x > y) or (y > z)
```

Logical **Or**

A	B	A or B
F	F	?
F	T	?
T	F	?
T	T	?

Logical Operators

```
x = 2
y = 1
z = 3
(x > y) or (y > z)
```

Logical **Or**

A	B	A or B
F	F	F
F	T	T
T	F	T
T	T	T

Order of Operations in Python

1) parentheses `(   )`
2) exponents `**`
3) positive (+) or negative (–) or **logical `not`**
4) multiplication (`*`) or division (`/`)
5) addition (+) or subtraction (–)
6) **relational operators (<,>,<=,>=,==,!=)**
7) **`and`**
8) **`or`**

Use parentheses, not order of operations

```
 (x > y) or  (y > z)  and (x > z)
((x > y) or  (y > z)) and (x > z)
 (x > y) or ((y > z)  and (x > z))
```

even if you don't need them, they add to readability

Best Practices

never assume you don't need them

Bitwise Operators

```
x = 0
y = 1
x & y        bitwise and
x | y        bitwise or
~x           bitwise not
x ^ y        bitwise xor
```

works with Boolean types or integers
(often, not always give same results)

logical operators "short circuit"
bitwise operators do not

String Types
String Operators

`StringsRegularExpressions.ipynb`

https://docs.python.org/3/tutorial/introduction.html#strings

https://docs.python.org/3/library/stdtypes.html#string-methods

strings

- strings contained within 'single quotes' or "double quotes"
- unlike some other languages, Python does not distinguish between characters and strings - a character is a string with one element

operations and functions applied to strings

- `len(s)`
  length of string

- `x = float("1.43")`
  convert from string

- ```
  a = "ABC"
  b = "def"
  print(a+b)
  ```

- `print(3*a)`

example of <u>operator overloading</u> in Python
(object-oriented language)
  ```

methods for string types (objects)

```
s = "THIS is A String"

print(s.lower())
print(s.split())
print(s.upper())
```

https://docs.python.org/3/library/stdtypes.html#string-methods

format method for strings

Perform a string formatting operation. The string on which this method is called can contain literal text or replacement fields delimited by braces `{}`. Each replacement field contains either the numeric index of a positional argument, or the name of a keyword argument. Returns a copy of the string where each replacement field is replaced with the string value of the corresponding argument.

- often used with `print()` function (to screen) or `write()` method (to file) but can be used anywhere with strings

```
a = 3
print("this num: {}".format(a))
s = "this num: {}"
s = s.format(a)
print(s)
```

https://docs.python.org/3/library/string.html#formatstrings

string literals / f-strings

more recent variant, like format strings but a bit more powerful and a bit cleaner

- also used with `print()` function (to screen) or `write()` method (to file) and can be used anywhere with strings

variables and expressions go directly within format string

```
a = 3
print(f"this num: {a}")

s = f"this num: {a}"
print(s)
```

https://realpython.com/python-f-strings/
https://docs.python.org/3/reference/lexical_analysis.html#literals

Homework 2 (Due Wed Sep 14th)
will ask you to implement some string literals
(f-string formatted strings)

string comparison

- basic string comparison is case-sensitive and fairly stupid

```
a = "Truck"
c = "truck"
print(a == c)
```

- determine is one string is part of another

```
a = "This is a string"
print("is" in a)
```

regular expressions in Python

- regular expressions define a powerful language for matching strings (here just some basics)

review slides and Jupyter notebook code (and web links) on regular expressions on your own
(they are a useful tool to know about)

regular expressions in Python

- regular expressions define a powerful language for matching strings (here just some basics)

```
pattern = r"S.*.dat"
fname = r"C:\Project\Data\S102-3.dat"
if (re.search(pattern, fname)):
    print("Success!")
else:
    print("Failure!")
```

https://jakevdp.github.io/WhirlwindTourOfPython/14-strings-and-regular-expressions.html

https://www.activestate.com/wp-content/uploads/2020/03/Python-RegEx-Cheatsheet.pdf

https://docs.python.org/3/library/re.html

https://docs.python.org/3/howto/regex.html#regex-howto

regular expressions in Python

```
re.search(r"^[a-z].*[3aj]x*z+Q$", r”X12345axxxzzzQ")
```

`^`	match expression to right at beginning of string
`.`	match any character
`[3aj]`	match any one of these characters
`*`	match 0 or more occurrences of the previous
`+`	match 1 or more occurrences of the previous
`$`	match express to left at end of string
`[a-z]`	match any one character in this range
`[0-9]`	match any one character in this range

regular expressions in Python

```
pattern = r"S-([0-9]*)-([0-9]*)-([a-zA-Z]*)"
fname = r"S-154-2-Control"
re.findall(pattern, fname)
```

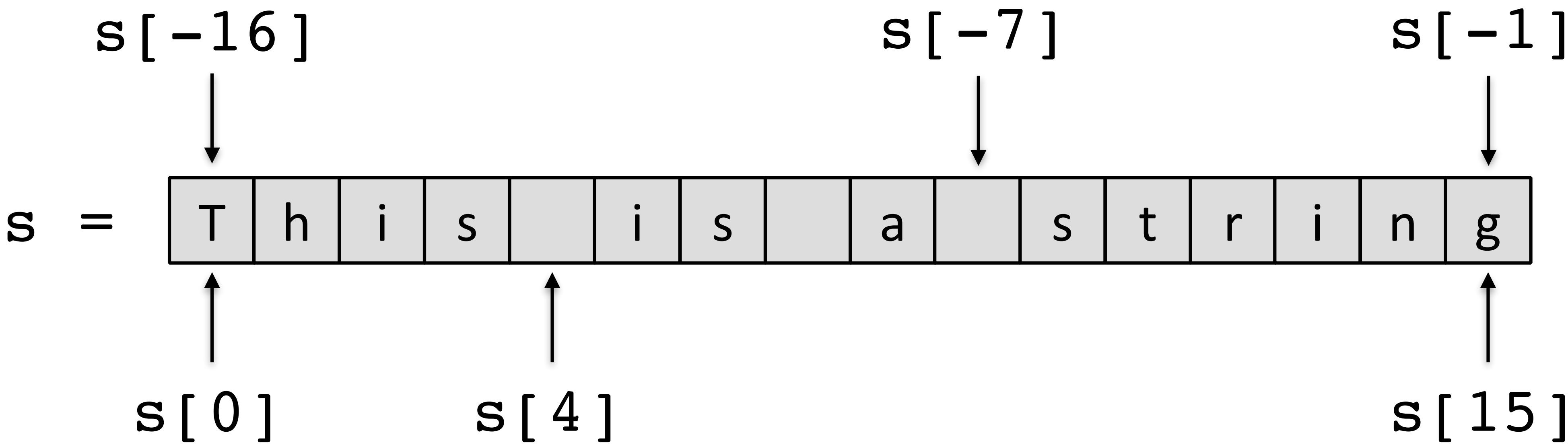
$$\texttt{"S-}\underbrace{\texttt{([0-9]*)}}_{\text{subject \#}}\texttt{-}\underbrace{\texttt{([0-9]*)}}_{\text{session \#}}\texttt{-}\underbrace{\texttt{([a-zA-Z]*)}}_{\text{condition}}\texttt{"}$$

string indices

- indices of strings (and lists, tuples, numpy arrays) start at 0, not at 1
- end of a string is index `len(s)-1`
- negative indices count from the end of a string

```
s = "This is a string"
print(s[0])          # first char
print(s[1])          # second char
print(s[len(s)-1])  # last char
print(s[-1])         # also last char
```

string indices



string slicing

```
s = "This is a string"
```

step (can step in a negative direction)

start-index

```
print(s[4:10:2])  ⟷  s[4]+s[6]+s[8]
```

end-index – 1
(doesn't include 10)

(doesn't include s[10])

stepping through each character in a string introducing **for loops** in Python

```
s = "This is a string"
```

`range()` returns a "sequence" of numbers from 0 to `len(s)-1`

```
for i in range(len(s)):
    print(i, "\t", s[i])
```

colon is necessary

indenting is necessary in Python
must be consistent through a program
convention is to use spaces

automatic in Jupyter Notebooks and IDEs

A Whirlwind Tour of Python, Jake VanderPlas
Chapter 8: Control Flow Statements

stepping through each character in a string introducing **for loops** in Python

```
s = "This is a string"
```

`range()` can have a start, end, and step

```
for i in range(0, len(s), 2):
    print(i, "\t", s[i])
```

a note on `range()`*

* in Python 3, `range()` returns a range object, which is iterable

`range()` does not actually create a list**

** it did in Python 2

```
N = 10 ** 15
```

larger than the memory of any personal computer

```
for i in range(N):
    if i >= 10:
        break
    print(i, end=', ')
```

*** replaces the default 'new line' with something else, here a `', '`

stepping through each character in a string introducing **for loops** in Python

```
s = "This is a string"
```

```
    can iterate over the string itself
for c in s:
    print(c)
```

strings are immutable

```
s = "This is a string"


# cannot change a string
s[3] = "X"   this throws an error
```

strings, tuples

immutable = unchangeable

vs.

mutable = changeable

lists, numpy arrays

(most variables in R are immutable)

Data Structures

Data Structures

one key to successful programming is
using the right kind of data structure and
using it the right way for the right problem

Why use a data structure?

imagine we have 10 subjects and each subject answers 10 true/false questions - 100 data points

we could create 100 variables, s1q1, s2q2, ... s2q1, s2q2, ... s10q9, s10q10

why do we use a data structure instead?

Why use a data structure?

imagine we have 10 subjects and each subject answers 10 true/false questions - 100 data points

we could create 100 variables, s1q1, s2q2, ... s2q1, s2q2, ... s10q9, s10q10

why do we use a data <u>structure</u> instead?

- access data more easily and more efficiently
- access data dynamically
- data are structured, systematically referenced
- all the data is “in the same place”

Data Structures in Python

- List
- Tuple
- Dictionary
- Sets

part of base Python

- Numpy Arrays
- Pandas

imported modules / packages

Python also easily support more sophisticated data structures (stacks, queues, trees, networks, etc.)

- we will use numpy arrays a lot in this course