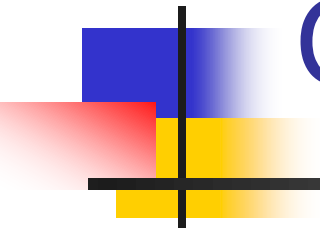
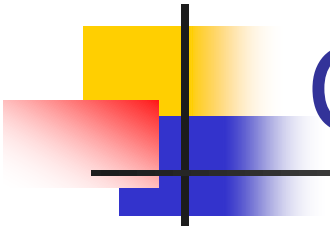


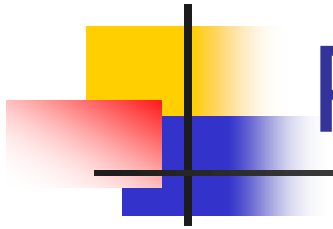
Computer Methods (MAE 3403)

Chapter 1
Introduction to Python



General information

- Recap of Python: not a comprehensive manual
- Refreshing your previous coursework on Python
- If you know another language, it is not difficult to pick up the rest as you learn



Python

- Object-oriented language developed in 1980s as a script language
  - Used widely now in engineering and computer science
  - Free, available on multiple all OS without mods
  - Easier to learn, more readable
- Not compiled code but interpreted (differences?)
  - Tested and debugged quickly compared to C and Fortran
  - Do not produce stand-alone applications
  - Need Python interpreter installed

Similarity to MATLAB

solve Ax = b via Gauss elimination

```
function x = gaussElimin(a,b)
n = length(b);
for k = 1:n-1
    for i= k+1:n
        if a(i,k) ~= 0
            lam = a(i,k)/a(k,k);
            a(i,k+1:n) = a(i,k+1:n) - lam*a(k,k+1:n);
            b(i)= b(i) - lam*b(k);
         end
     end
end
for k = n:-1:1
    b(k) = (b(k) - a(k,k+1:n)*b(k+1:n))/a(k,k);
end
x = b;
```

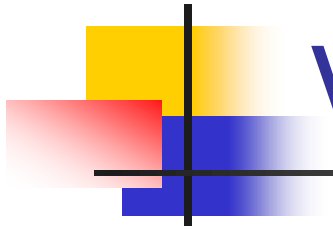
```
from numpy import dot
def gaussElimin(a,b):
    n = len(b)
    for k in range(0,n-1):
        for i in range(k+1,n):
            if a[i,k] != 0.0:
                lam = a [i,k]/a[k,k]
                a[i,k+1:n] = a[i,k+1:n] - lam*a[k,k+1:n]
                b[i] = b[i] - lam*b[k]
    for k in range(n-1,-1,-1):
        b[k] = (b[k] - dot(a[k,k+1:n],b[k+1:n]))/a[k,k]
    return b
```

Obtaining Python

- www.python.org/getit
- Or install Anaconda

- Some needed extension modules: scipy, numpy, matplotlib, etc.

- https://docs.python.org/3/tutorial



Variables and assignment

- Variables: a value of a given type stored in a fixed memory location
- Variable names: letters, numbers, underscores, the first character must be a letter or underscores
  - dist vs. x, nRabbits vs. y
- x = 1: takes the known value 1, assigns that value to a variable with name ‘x’.

Assignment

- The equal sign '=' is different from a truth statement (e.g., x equals 2), e.g., x = x + 1
- Value and type may be changed dynamically

```
>>> b = 2          # b is integer type
>>> print(b)
2
>>> b = b*2.0   # Now b is float type
>>> print(b)
4.0
```

b is changed from an integer to a floating number.

- del x: clear variable x from the workspace

Type conversions

```
>>> a = 5
>>> b = -3.6
>>> d = '4.0'
>>> print(a + b)
1.4
>>> print(int(b))
-3
>>> print(complex(a,b))
(5-3.6j)
>>> print(float(d))
4.0
>>> print(int(d))  # This fails: d is a string
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    print(int(d))
ValueError: invalid literal for int() with base 10: '4.0'
```

truncation

`int(a)`	Converts a to integer
`float(a)`	Converts a to floating point
`complex(a)`	Converts to complex $a + 0j$
`complex(a,b)`	Converts to complex $a + bj$

Data types: Strings

- an array of characters enclosed in single or double quotes: w = “Hello World”
- String: indices to indicate the location of each character

Character	H	e	l	l	o		W	o	r	l	d
Index	0	1	2	3	4	5	6	7	8	9	10

- w[6]? w[6:11] (slicing)?

String operations

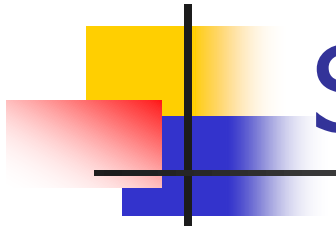
- More on slicing: [start:end:step]
  - w[6:]: slice to the end, w[:5]: slice from the beginning
  - w[::2]: every other character
  - Negative index: counting from the end, w[6:-2]
- Concatenation: +

```
>>> string1 = 'Press return to exit'
>>> string2 = 'the program'
>>> print(string1 + ' ' + string2)  # Concatenation
Press return to exit the program
>>> print(string1[0:12])           # Slicing
Press return
```

More operations

- `don’t’ -> w = ‘don\’t’
- str(1) becomes a string ‘1’
- w.upper(): turns to upper case
- w.count(‘a’): count the number of occurrence of ‘a’
- w.replace(‘a’,’b’): replace ‘a’ in w by ‘b’
- len(w): length of the string w
- split:

```
>>> s = '3 9 81'
>>> print(s.split())    # Delimiter is white space
['3', '9', '81']
```



String

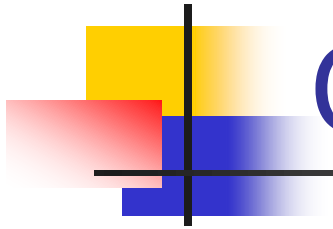
- String is an immutable object
  - individual characters cannot be modified with an assignment statement; it has a fixed length

```
>>> s = 'Press return to exit'
>>> s[0] = 'p'
Traceback (most recent call last):
  File ''<pyshell#1>'', line 1, in ?
    s[0] = 'p'
TypeError: object doesn't support item assignment
```

Tuples

- A sequence of *arbitrary objects* separated by commas and enclosed in parenthesis
  - Single object: `x = (2,)`
- Supports the same operations as strings, also immutable

```
>>> rec = ('Smith','John',(6,23,68))    # This is a tuple
>>> lastName,firstName,birthdate = rec  # Unpacking the tuple
>>> print(firstName)
John
>>> birthYear = birthdate[2]
>>> print(birthYear)
68
>>> name = rec[1] + ' ' + rec[0]
>>> print(name)
John Smith
>>> print(rec[0:2])
('Smith', 'John')
```



Operations

- tuple_1 = (1,2,3,2)
- len(tuple_1)
- tuple_1.count(2)
- unpacking: a,b,c,d=tuple_1
  - there are as many variables on the left as there are on the right

List

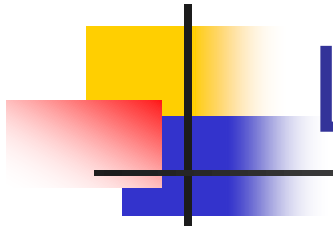
- Similar to tuple, but mutable. Enclosed by brackets

```
>>> a = [1.0, 2.0, 3.0]       # Create a list
>>> a.append(4.0)             # Append 4.0 to list
>>> print(a)
[1.0, 2.0, 3.0, 4.0]
>>> a.insert(0,0.0)           # Insert 0.0 in position 0
>>> print(a)
[0.0, 1.0, 2.0, 3.0, 4.0]
>>> print(len(a))             # Determine length of list
5
>>> a[2:4] = [1.0, 1.0, 1.0] # Modify selected elements
>>> print(a)
[0.0, 1.0, 1.0, 1.0, 1.0, 4.0]
```

Matrix: nested lists

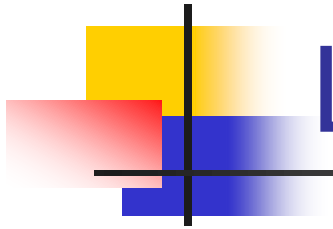
```
>>> a = [[1, 2, 3], \
         [4, 5, 6], \
         [7, 8, 9]]
>>> print(a[1])          # Print second row (element 1)
[4, 5, 6]
>>> print(a[1][2])       # Print third element of second row
6
```

- \: continuation character
- Indeed, we use *array (from numpy)* more often than *list* to represent matrices.



List operations

- list_1 = [1, 2, 3]
- list_2 = ['Hello', 'World']
- Adding lists: list_1 + list_2
- append: list_1.append(4)
- insert: list_1.insert(2,'center')
- delete an item: del list_1[2]
- Check an item: 3 **in** list_1
- empty list: list_5 = [], list_5.append(5)



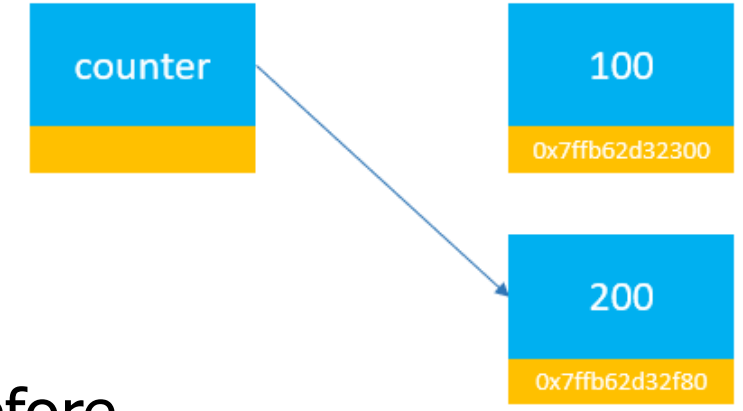
List vs. tuple

- Tuples are immutable and usually contain heterogeneous sequence of elements that are accessed via unpacking
  - [('apple', 3), ('banana', 4) , ('orange', 1), ('pear', 4)]
- Lists are mutable and usually contain homogeneous elements accessed by iterating over the list
  - [‘apple’, ‘banana’, ‘orange’, ‘pear’]

Immutable vs. mutable objects

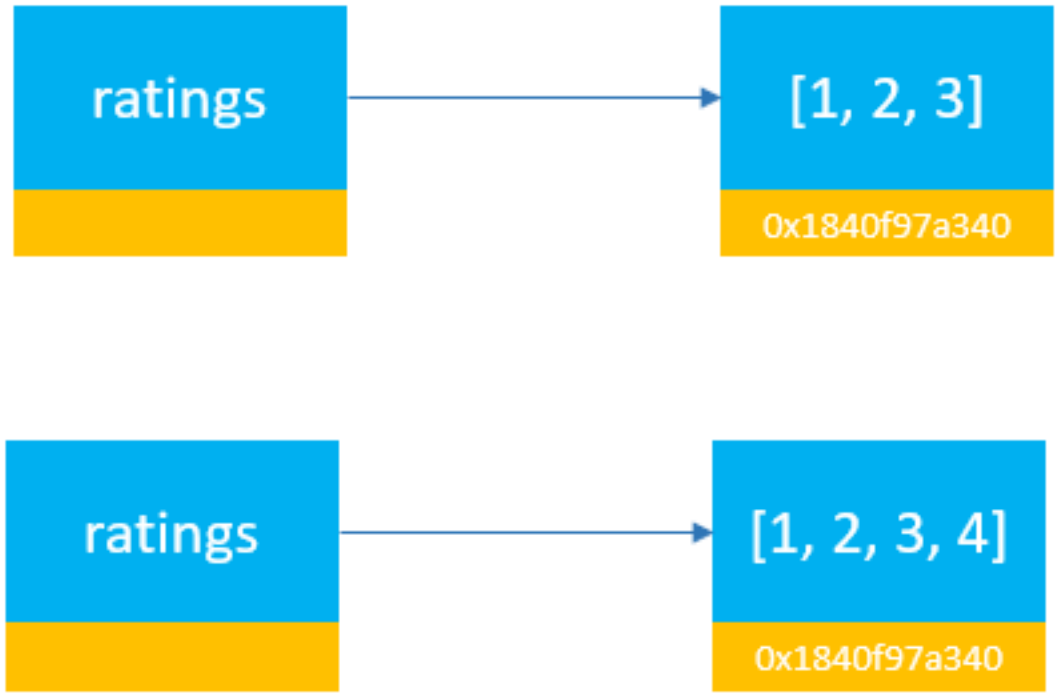
- immutable objects: numbers, strings, tuples,…
- mutable objects: lists, dictionaries, sets,…
- immutable: reassignment doesn't change the value of the object. Python creates a new integer object and reassigns the *counter* to reference the new object

```
counter = 100
print(id(counter)) # memory address of counter
print(hex(id(counter))) # in hexadecimal
counter = 200
print(hex(id(counter))) # expect to be different from before
```



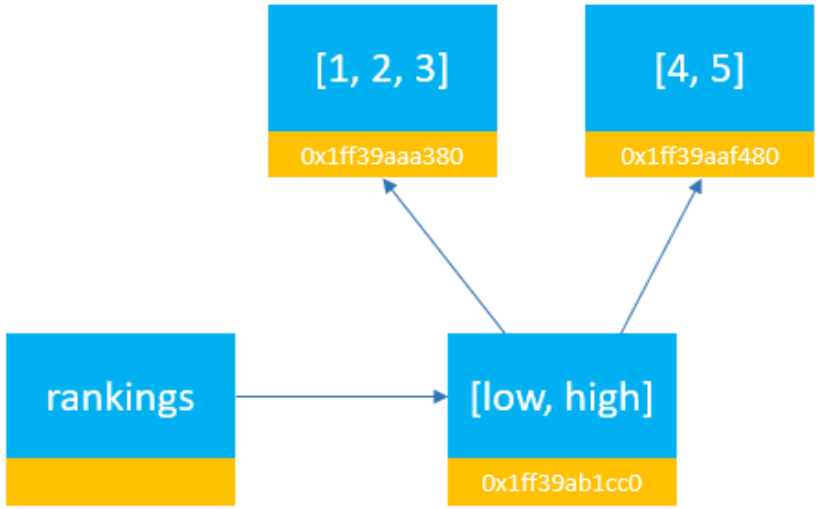
Mutable

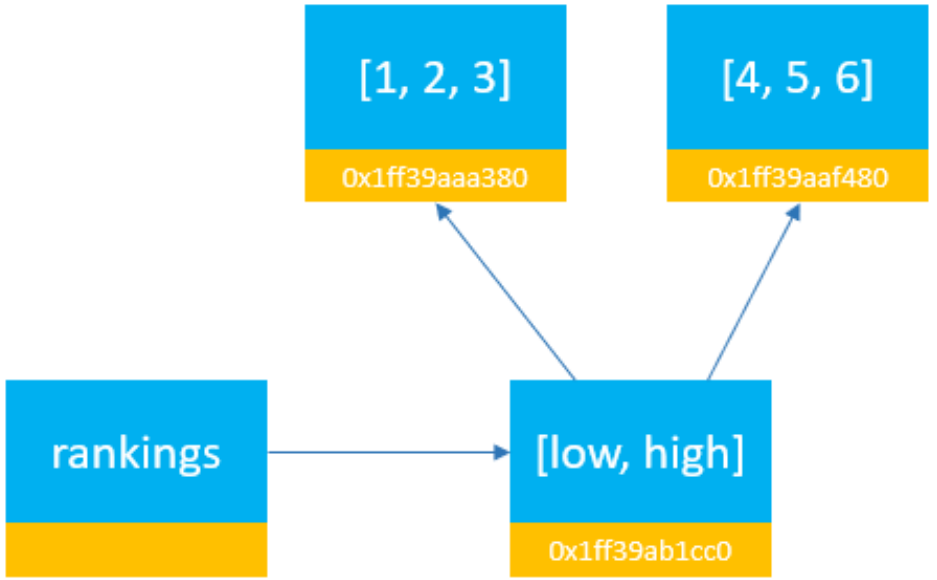
- ratings = [1,2,3]
- print(hex(id(ratings)))
- ratings.append(4)
- print(hex(id(ratings)))



immutable containing mutable objects

- low = [1,2,3] high = [4,5] rankings = (low,high) # this is a tuple
- high.append(6)

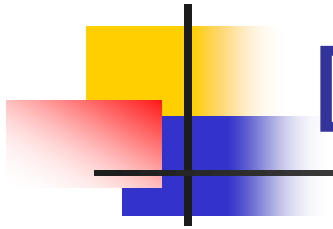




Possible confusion

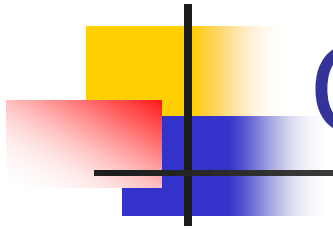
- If *a* is a mutable object, *b = a* does not create a new object b, but creates a new reference (pointer) to *a*.
- To create an independent copy of a list *a*, use *c = a[:]*.

```
>>> a = [1.0, 2.0, 3.0]
>>> b = a                  # 'b' is an alias of 'a'
>>> b[0] = 5.0             # Change 'b'
>>> print(a)
[5.0, 2.0, 3.0]            # The change is reflected in 'a'
>>> c = a[:]               # 'c' is an independent copy of 'a'
>>> c[0] = 1.0             # Change 'c'
>>> print(a)
[5.0, 2.0, 3.0]            # 'a' is not affected by the change
```



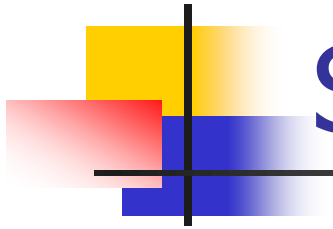
Dictionaries

- Key-value pairs: each key maps to a corresponding value, defined by a pair of braces {}, while elements are a list of comma-separated key:value pairs
  - dict_1 = {'apple':3, 'oragne':4, 'pear':2}
- Indexed by keys, accessed by keys: dict_1['apple']
- dict_1.keys(), dict_1.values(), len(dict_1), dict_1.items()
- Keys can be any immutable type (strings/numbers)



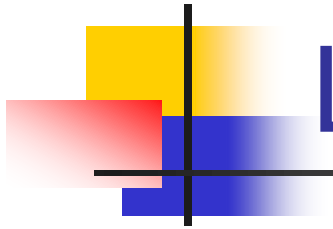
Operations

- school_dict = {}
  school_dict['UC Berkeley'] = 'USA'
  school_dict['Oxford'] = 'UK'
- Convert a list of tuples:
  dict([("UC Berkeley", "USA"), ('Oxford', 'UK')])
- "UC Berkeley" **in** school_dict
- list(school_dict): turns the dictionary to a list of keys



Sets

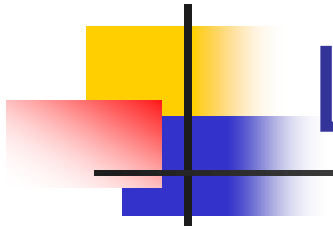
- an unordered collection with no duplicate elements.
- Supports mathematical operations like union, intersection, difference, etc.
- Defined by {}, elements separated by commas
- set_1 = set([1, 2, 2, 3, 2, 1, 2]), set('Banana')
- set_1.union(set_2), set_1.intersection(set_2), set1.issubset(set_2), …



Last time: difference between

a = b

a = b[:]



Last time: difference between

- Shallow copy

  a = b

- Deep copy

  a = b[:]

Operations

+	Addition
−	Subtraction
*	Multiplication
/	Division
**	Exponentiation
%	Modular division

Arithmetic

`a += b`	`a = a + b`
`a -= b`	`a = a - b`
`a *= b`	`a = a*b`
`a /= b`	`a = a/b`
`a **= b`	`a = a**b`
`a %= b`	`a = a%b`

<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Comparison

Examples

```
>>> s = 'Hello '
>>> t = 'to you'
>>> a = [1, 2, 3]
>>> print(3*s)                  # Repetition
Hello Hello Hello
>>> print(3*a)                  # Repetition
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> print(a + [4, 5])           # Append elements
[1, 2, 3, 4, 5]
>>> print(s + t)                # Concatenation
Hello to you
>>> print(3 + s)                # This addition makes no sense
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    print(3 + s)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Logical expressions and operators

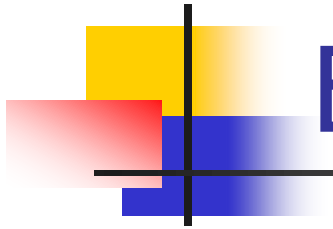
- Boolean variable: *true (=1)* and *false (=0)*
- *3<4: true, 3>4: false*
- Logical operators: and, or, not

Operator	Example	Results
and	P and Q	True if both P and Q are True False otherwise
or	P or Q	True if either P or Q is True False otherwise
not	not P	True if P is False False if P is True

Examples

```
>>> a = 2              # Integer
>>> b = 1.99           # Floating point
>>> c = '2'            # String
>>> print(a > b)
True

>>> print(a == c)
False
>>> print((a > b) and (a != c))
True
>>> print((a > b) or (a == b))
True
```



Examples: Logical expressions

- (1 and not 1) or (1 and 1)
- (3 > 2) + (5>4)
- 1+3 > 2 + 5
- (1+3) > (2+5)

Conditionals

```
if condition:
    block
```

```
elif condition:
    block
```

⋮

```
elif condition:
    block
```

```
else:
    block
```

```
def sign_of_a(a):
    if a < 0.0:
        sign = 'negative'
    elif a > 0.0:
        sign = 'positive'
    else:
        sign = 'zero'
    return sign

a = 1.5
print('a is ' + sign_of_a(a))
```

Ternary operators

- one-line code to evaluate the first expression if the condition is true, otherwise it evaluates the second expression:

```
expression_if_true if condition else expression_if_false
```

```
is_student = True
person = 'student' if is_student else 'not student'
print(person)
```

- Makes code more concise, commonly used in list

Loops

while loop

```
while condition:
    block
```

```
else:
    block
```

```
nMax = 5
n = 1
a = []                  # Create empty list
while n < nMax:
    a.append(1.0/n)     # Append element to list
    n = n + 1
print(a)
```

for loop

```
for target in sequence:
    block
```

```
nMax = 5
a = []
for n in range(1,nMax):
    a.append(1.0/n)
print(a)
```

Looping techniques

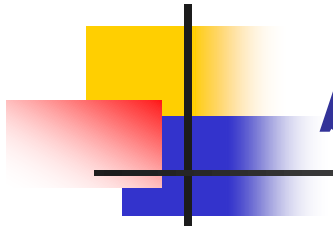
- looping through a list: `for k in list:`
- range(n): [0,1,…,n-1], so you can use `for k in range(n):`
- Iterate position index and corresponding value of a list
  - `for k, v in enumerate(list):` or `for k, v in enumerate(['tic','tac','toe']):`
- Loop a sequence/list in reverse order:
  - `for k in reversed(range(1,10,2)):`
- Loop through a sorted order of list: `for k in sorted(list)`

Looping techniques

- Loop two or more sequences/lists: use zip

- Loop through dictionaries:

```
knights = {'gallahad': 'the pure', 'robin': 'the brave'}
for k, v in knights.items():
    print(k,v)
```



break and *continue*

- break: terminate any loop. will not run *else*
- continue: skip a portion of the loop. Immediately returns to the beginning of the loop without executing statements below *continue*

Examples

```
list = ['Jack', 'Jill', 'Tim', 'Dave']
name = eval(input('Type a name: '))  # Python input prompt
for i in range(len(list)):
    if list[i] == name:
        print(name,'is number',i + 1,'on the list')
        break
else:
    print(name,'is not on the list')
```

```
Type a name: 'Tim'
Tim is number 3 on the list

Type a name: 'June'
June is not on the list
```

```
x = []                    # Create an empty list
for i in range(1,100):
  if i%7 != 0: continue   # If not divisible by 7, skip rest of loop
  x.append(i)             # Append i to the list
print(x)
```

Comprehensions

- A way to do iterations: list (dictionary, set) comprehensions
- List comprehensions:
  [Output Input_sequence Conditions]
-
```
x = range(5)
y = [i**2 for i in x]
print(y)
```

-
```
x = range(5)
y = []
for i in x:
    y.append(i**2)
print(y)
```

More examples

- `y = [i**2 for i in x if i%2 == 0] print(y)`

- `y = [i + j for i in range(5) for j in range(2)] print(y)`

```
y = []
for i in range(5):
        for j in range(2):
                y.append(i + j)
print(y)
```

Dictionary comprehension

- ```
  x = {'a': 1, 'b': 2, 'c': 3}
  {key:v**3 for (key, v) in x.items()}
  ```
  ```

Core math functions

`abs(a)`	Absolute value of `a`
`max(`*sequence*`)`	Largest element of *sequence*
`min(`*sequence*`)`	Smallest element of *sequence*
`round(a,n)`	Round `a` to `n` decimal places
`cmp(a,b)`	Returns $\begin{cases} -1 \text{ if } a < b \\ 0 \text{ if } a = b \\ 1 \text{ if } a > b \end{cases}$

- Other math functions available in the *math* module

Input

- *input(*prompt*)*: accept user input. Displays the *prompt* and reads a line of input converted to a *string.*
- *eval(string):* convert the string to a numerical value

```
a = input('Input a: ')
print(a, type(a))           # Print a and its type
b = eval(a)
print(b,type(b))            # Print b and its type
```

```
Input a: 10.0
10.0 <class 'str'>
10.0 <class 'float'>

Input a: 11**2
11**2 <class 'str'>
121 <class 'int'>
```

- *a = eval(input(*prompt*))*

Output

- *print(*obj1, obj2, …): convert obj1, obj2, .. to strings and print them on the same line, separated by space.
- *newline:* \n

```
>>> a = 1234.56789
>>> b = [2, 4, 6, 8]
>>> print(a,b)
1234.56789 [2, 4, 6, 8]
>>> print('a =',a, '\nb =',b)
a = 1234.56789
b = [2, 4, 6, 8]
```

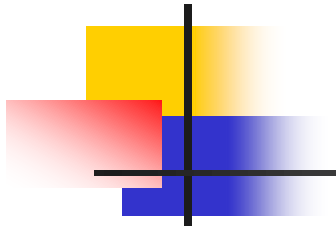
formatted output

'{:*fmt1*}{:*fmt2*}...'.format(*arg1*,*arg2*,...)

- fmt1, fmt2, …, are the format specs for arg1, arg2, …

*w*d	Integer
w.*d*f	Floating point notation
w.*d*e	Exponential notation

- w: width of the field, d: the number of digits after the decimal point.



```
>>> a = 1234.56789
>>> n = 9876
>>> print('{:7.2f}'.format(a))
1234.57
>>> print('n = {:6d}'.format(n))    # Pad with spaces
n =  9876
>>> print('n = {:06d}'.format(n))  # Pad with zeros
n =009876
>>> print('{:12.4e} {:6d}'.format(a,n))
  1.2346e+03   9876
```

Advanced print

- Add an r before the string. The r represents raw and will render the text literally:

```
print(r"Now the string is raw! \n \r")
```

- Print f-string

```
my_float = 444.44445
print(f'My float: {my_float:010.3f}')
```

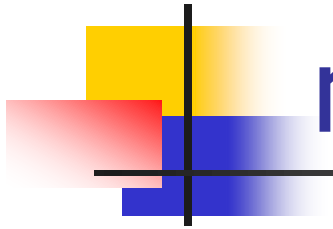
https://the-examples-book.com/programming-languages/python/printing-and-f-strings

Example: matrix multiplication

- Multiply matrices a and b, and save the result to c
  - Check the dimensions (how?)
    - no. of cols of a should equal to no. of rows of b
  - Get the dimension (size) of the product c (how?)
  - Initialize a list c (how?)
  - For the i,jth element of c: #how to obtain c[i][j]?

$$c[i][j] = a[i][0]b[0][j] + a[i][1]b[1][j] + \ldots + a[i][ncola-1]b[ncola-1][j]$$

    - Requires a summation. How?



mini-Quiz

- Given a = 1+ (3>2) + 5, what is the value of a?

- Write a logical expression to determine if a fortnight (2 weeks) is longer than 100,000 seconds. In other words, if a fortnight is longer, the expression should evaluate to True. Otherwise, it should evaluate to False.