Python data types and structures

Numbers

Integers: for example −1, 0, 2 and 1,017,347

Integers are whole numbers and are exact.

Floats: for example −1.1, 0.1, 3.14 and 2.99e10

Floats are not exact because computers cannot generally represent them exactly.

So: $\frac{1}{N} * N$ is not always exactly 1.

It might be 0.99999999999... or 1.00000000004

These are both very close to but not equal to 1.0.

Floats are **fluffy.**

Complex numbers

You know how $\sqrt{-1} = i$?

in Python we represent that as:

```
complex(0, 1)
```

the real part

the imaginary part

if `z = complex(0, 1)`

```
z.real = 0
z.imag = 1
```

Complex numbers are "complex" and we won't use them very often. There is a whole cmath library and special functions in numpy for working with them.

"Strings" are alphanumeric characters including white space characters like TAB, space, etc... in quotes.

A single-line string (has no line break, new line or carriage return)

can be 'single quoted' or "double quoted".

You can nest quotes like:

"a 'single quote' in a double quote"

OR

'a "double quote" in a single quote'

Some characters are special.

\n = newline
\t = tab
\r = carriage return
\b = backspace
\f = form feed
\' = single quote
\" = double quote
\\ = back slash

\a = bell sound
\NNN = octal value
\xNNN = hex value
\v = vertical tab

"Special" means you don't literally get those characters when you print them!

"a\tb" ⇒ "a b"

a tab character

So, if you literally mean to use the special characters you have these options:

① Escape them, so to get "a\tb" printed use \\ in the string to represent a back slash.

"a\\tb" ⇒ "a\tb"

② Use a "raw" string by prefixing the quoted string with an r:

r"a\tb" ⇒ "a\tb"

in a raw string no special characters are interpreted. Raw strings are very convenient for holding LaTeX commands (which always start with \).

"""Multiline strings are triple quoted"""

"""These strings can also use triple "double quotes""""

Formatted f-strings

f-strings are strings with an f before the first quote, like f'a string'

You can put variables in an f-string and when you print it the value of the variable will get inserted where you specify, in the format you say. The syntax is:

{var : fmt}

For example:

a = 1.2

f'the var a = {a:1.1f}' ⇒ 'the var a = 1.2'

- var → a
- fmt → 1.1f
  - f → float
  - w.df
    - d → decimal places
    - w → width in characters

You can also use scientific notation (e, E, g or G).

An f-string can have many variables in it, and can be used in single and triple-quoted strings.

⇒ A thread.

Lists are one kind of container for data.

A = [item0, item1, milk, pencils]

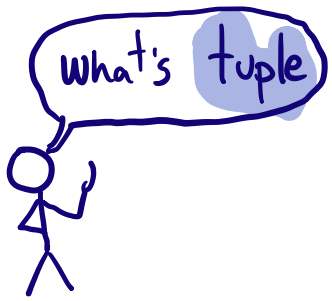
A list can contain many different kinds of items. They do not need to be the same kind of item. They can even hold other lists!

A list is surrounded by square brackets [ ].

Lists can be changed after they are made.

so: A[0] = item2 ⇒ [item2, item1, milk, pencils]

This is called **mutability**. Use indexing to access list items: A[2] ⇒ milk



B = (item0, item1, milk)

A tuple is also a container for data, and it can also contain many different kinds of data.

A tuple is surrounded by parentheses ( ).

Note, if a tuple only has one item, it must also have a single comma like

(singleItem,)

to differentiate it from an expression like (singleItem) which simply evaluates to singleItem.

Tuples CANNOT be modified after you make them

B[0] = item2 ⇒ Error, this is not allowed!

Immutability is useful when you want to make sure that the contents cannot be changed.

You use indexing to get tuple items.

B[0] => item0

Dictionary

A dictionary is a data structure that stores key: value pairs. The keys can be any "hashable" data type

- integers
- strings
- tuples

} common choices for keys

Lists and arrays are not hashable (because they are mutable) so they cannot be used as keys. They can be values though.

```
D = {key1: value1,
     key2: value2,
       ⋮
     keyn: valuen}
```

Each key is separated from the value with a colon. Each key: value pair is separated by a comma.

The whole dictionary is inside curly brackets {}

You access values by indexing with a key

D[key1] ⇒ value1

Numpy arrays

A numpy array is *like* a list that you can do math on.

```
N = np.array([item0, item1, ... itemn])
```

This is basically a list as an argument to the np.array function

arrays are *usually* all the same types like integers and floats. It is possible for them to be object arrays that have different kinds of objects in them.

Arrays provide a simple syntax for elementwise and matrix math.

e.g.
- `N*N` ⇒ elementwise multiplication
- `N@N` ⇒ matrix multiplication.
- `f(N)` ⇒ elementwise application of f on N.

Sets

Sets are a container for unique items.

- They are unordered, and the order may change when you use them
- They cannot be indexed (since they are unordered)
- They cannot have duplicate values
  - duplicates are eliminated.
- A set can contain different data types

Sets are defined in curly brackets (similar to dictionaries, but no key: value pairs, just elements)

S = {'a', 'b', 'b'} => {'a', 'b'}

this duplicate is eliminated

Sets are most useful for getting a unique set, and for doing set operations like union and intersection. We won't see sets very often in 06-623.