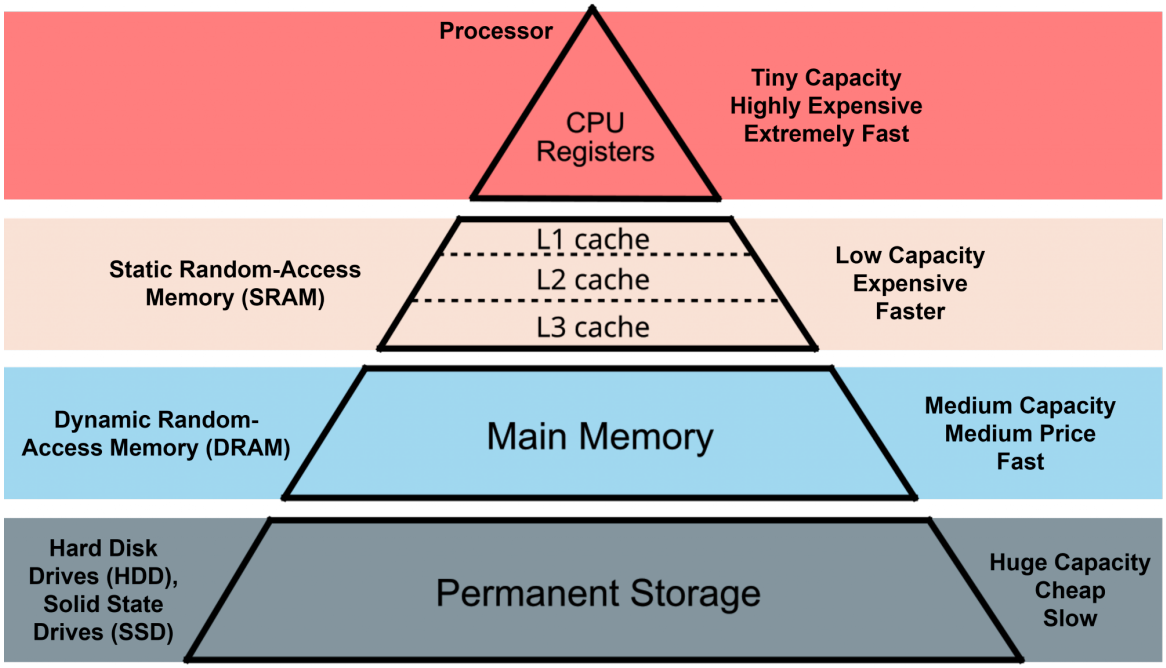
This lecture requires the file books.csv.

Memory and bits

Last time, we began to create values that could be stored as variables. But how many values like this can we store? What's the limit?

Data takes up memory. In a file, data takes up space on your hard drive. While a program is running, the program and its variables take up space in RAM, the computer's working memory. If the data is being actively manipulated by the program, it may take up some of the limited space in the CPU's caches or in its registers – the CPU's workspaces for operating on data.

The CPU can access data in its own caches and registers the fastest. Next fastest is RAM. It takes more time to fetch data from a file. And it takes still more time to download data from the Internet.



https://spear-itn.eu/memory-hierarchy-how-does-computer-memory-work/

Hard drive and cloud storage are cheap enough that we usually don't have to worry about them as finite resources, but you can still run out of RAM, which will either crash the program or start making your computer act funny. But Python is nice in that it takes some automatic steps to ensure that memory that isn't being used anymore is freed up for other purposes ("garbage collection").

In all these cases, data is represented in binary - a code of 1's and 0's. Numbers like *5* are represented in memory by their binary form, *00000101*. (That's $1 * 2^2 + 0 * 2^1 + 1 * 2^0$.) Strings

are typically represented by one *byte* (eight bits) per character - the code for a lowercase "a", for example, is *01100001*.

When running Python code, the Python interpreter will keep track of the meaning of all that binary - the interpretation of some bits depends on the data's *type*.

Strings

Strings like "Hello, World!" are text. Strings are enclosed in single or double quotes. The + operator *concatenates* two strings, gluing them together to produce a new string. You can't concatenate numbers with strings unless you convert the number to a string with the str() function.

```
coursename = "DS110"
compliment = " is great!"
print(coursename + compliment)
print(str(5) + " star course") # Remove str() and we get an error
```

```
DS110 is great!
5 star course
```

Strings that don't have any emoji or similarly unusual characters use one *byte* of memory per character. Shakespeare's play *Hamlet* has 165,000 characters, so it takes 0.165 megabytes - not much compared to the several *giga*bytes of RAM available on modern laptops.

Integers

Integers in Python are exact and can be arbitrarily large. Here we calculate 2^{1000} without breaking a sweat.

```
print(2 ** 1000)
```

```
10715086071862673209484250490600018105614048117055336074437503883703510511249361224931983788
```

Integers are represented in binary, where every bit represents a different power of 2. We can see this binary if we call the bin() function. (This is not usually necessary, but gives you a sense of what is actually being stored in memory.) Very big numbers take up more space.

```python
print(bin(5))  # 5 = 4 + 1 = 1*2^2 + 0*2^1 + 1*2^0
```

```
0b101
```

```python
print(bin(2**1000))
```

```
0b1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Float

This may be the most peculiar of the basic Python data types. Floats are Python's data type for representing numbers with digits after the decimal point: 0.5, 1.1, -2.34212411, and so on. But the way they're represented in binary leads to the occasional inaccuracy.

```python
0.1 + 0.1 + 0.1
```

```
0.30000000000000004
```

This happens because numbers can't always be represented exactly in the float binary representation. The binary consists of a sign bit, binary for an exponent, and binary for the "significand." The value of a float is then $(-1)^{sign} * Significand * 2^{Exponent}$. This works great if you're trying to represent "0.25" with sign=0, significand = 1, exponent = -2, but turns out to have no choice of values that represents 0.1 exactly. It's not all bad - floats are generally accurate to about 15 digits of precision - but you can expect them to very often have these little inexact parts.

If you're not sure whether a number is a float or an integer, look for a digit after the decimal point. 2.0 is a float, but 2 is an integer.

Boolean

A Boolean value is either True or False. When an inequality like "$2 < 5$" is evaluated in Python, the value that is produced is a Boolean.

```python
a = 2 < 5
print(a)
```

```
True
```

Other inequalities that are good to know about are <= and >= for "less than or equal" and "greater than or equal", == for "equal", != for "not equal". Using these to compare values results in a true or false Boolean.

```
print(5<=5)
print(5==5)
print(5!=5)
```

```
True
True
False
```

While it would make sense for a boolean to take 1 bit to encode - 0 for False, 1 for True - in practice, every object in Python has some additional information encoded about it, and this brings booleans to 28 bytes. (Strings, integers, and floats similarly use some extra memory beyond what we described.) In many other languages, booleans take a byte and are 00000000 (False) or 00000001 (True).

Other types - objects in general

Python has many types besides integer, string, float, and boolean - those are just the most important ones. Another type is DataFrame, an instance of which we created at the end of last lecture - relatively complicated as types go. All types have some data that is associated with them, and most also have functions specific to them called *methods*. We'll explore these more later.

Checking types

The type() function will tell us what type of data is stored in a variable if we're not sure.

```
my_string = "example"
print(type(my_string))
my_int = 64
print(type(my_int))
# Upload books.csv before running this code
from pandas import read_csv
df = read_csv('books.csv')
print(type(df))
```

```
<class 'str'>
<class 'int'>
<class 'pandas.core.frame.DataFrame'>
```

You can also see the types of variables if you use the variable inspector (bottom left of the Colab screen).

Getting input and type conversion

Here's an example of a place where type matters. The input() function gets input from the user (its argument is the prompt displayed to the user). It always returns a string. That works if we just want to print what the user typed, but if it's a number we want to do math to, we need to call a function like int() to convert it to a number.

```
year = input('What year is it?  ')
print("Oh, it's " + year + "...") # String + string concat - ok
next_year = int(year) + 1 # Because we want to actually add
print("Next year is " + str(next_year)) # Convert back to string
```

```
What year is it?  2026
Oh, it's 2026...
Next year is 2027
```

Exercise

Try writing some code that asks for one number, then another number, then reports the product (*) of the two numbers.

```
n1 = input('First number?  ')
n2 = input('Second number?  ')
my_prod = int(n1) * int(n2)
print("Product is " + str(my_prod))
```

```
First number?  10
Second number?  20
Product is 200
```

Even though we're not 100% teaching DataFrames just yet, this is how you can inspect the types of each column of a DataFrame. It says "object" when it means string and reports the number of bits used for the numbers, which is 64 bits each.

```
print(df.dtypes) # dtypes is an attribute of df - we'll cover this later
```

```
book_id                        int64
goodreads_book_id              int64
best_book_id                   int64
work_id                        int64
books_count                    int64
isbn                          object
isbn13                       float64
authors                       object
original_publication_year    float64
original_title                object
title                         object
language_code                 object
average_rating               float64
ratings_count                  int64
work_ratings_count             int64
work_text_reviews_count        int64
ratings_1                      int64
ratings_2                      int64
ratings_3                      int64
ratings_4                      int64
ratings_5                      int64
image_url                     object
small_image_url               object
dtype: object
```

Question

Explain why each of these columns has the type it does: average_rating, ratings_count, image_url.

Scales of memory

- A bit can be 0 or 1. Usually we don't manipulate single bits directly in Python.
- A byte is 8 bits, with $2^8 = 256$ different possible values. A byte can represent a character in text, or a single pixel's brightness in a black and white image.
- A token is a meaningful piece of text like "math" or "code" or "-ish", and is therefore a few bytes long. AIs break messages down into tokens and charge money by the token.

- 64 bits is the size of a register on a typical modern CPU. Operations like addition are fastest when they combine values that take 64 bits or less to represent.
- Every line of code we write is about 25 characters on average, thus 25 bytes per line. Most programs you write in this course will require fewer than 100 lines of code, or 2.5 KB.
- The complete works of Shakespeare consist of 3.5 million characters, and therefore 3.5 MB. This is also roughly the size of a song in mp3 format, or a single uncompressed 1024×1024 color photo, or the dataset of 10,000 books we examined last time.

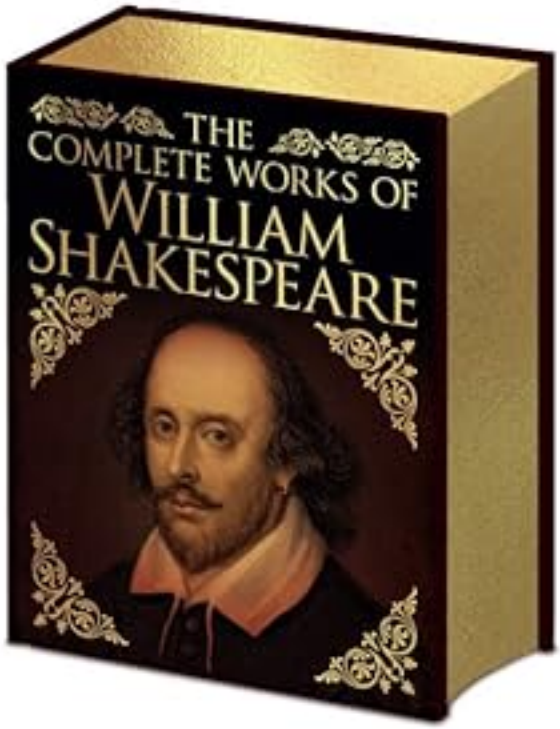


Figure 1: image.png

Figure 2: image.png

1024 x 1024 bird

- If you have a new computer, you probably have about 16-32 GB of RAM - enough to store all the text of a small library. (Or a few programs running simultaneously - Chrome takes 4-8GB.)
- A new hard drive may be 1 TB in size. That's a million MB (1000 GB), or enough storage

for an mp3 playlist that lasts until the middle of next year without repeats. This is also roughly the scale of the data used to train ChatGPT when it first came out (version 3.5).

- An exabyte (EB) is 1 million TB - this is roughly the scale of all the videos stored on YouTube. If a single 1TB hard drive is like a square mile, an exabyte is like the area of the United States.
- One zettabyte is 1,000 EB. It's estimated that all the data stored in hard drives around the world is on the scale of a hundred zettabytes.

Exercise: Memory

Find a file on your laptop, tell us what it is, and tell us the file size (Mac right-click on the file -> Get Info, Windows right-click -> Properties).

Important points from today

- Some important data types in Python are string (text), integer (exact!), float (not exact!), and booleans (True/False).
- Some operations, like +, behave differently when used with different types (concatenation versus addition).
- All the types are represented with binary. Values that require more binary take more space in memory and on a hard drive.