Warmup (L11)

- You have files named, e.g., `"2023-04-13_finances.txt"`. Write functions that take a filename as argument to extract info:
    - `fileYear(filename)` returns the year as an int
    - `fileMonth(filename)` returns the month as an int
    - `fileDay(filename)` returns the day of the month as an int
    - `fileSubject(filename)` returns the subject ("finances" above) as a string

Sorting and Dictionaries

M5

in order? put things Why

CS114 M5

Sorting

- Lists can be in any order (whatever order you put things in it)
- Some things would be easier if they were in order: What’s the least value? The most? The biggest gap?
- So, in some cases it’s useful to put things in order

Sorting a list

- Lists have a method to do this!

```
lst = [2, 4, 6, 0, 1]
lst.sort()
print(lst) # [0, 1, 2, 4, 6]
```

Sorting a list

- Lists have a method to do this!

```
lst = [2, 4, 6, 0, 1]
x = lst
lst.sort()
print(x) # [0, 1, 2, 4, 6]
```

This sorts *in place*, so all references will see the same sorting.

It’s just <

- Under the surface, it’s just using < to sort
- < works on numbers and strings, but not *between* numbers and strings

```
lst = [99, "bottles of beer on the wall"]
lst.sort() # Error!
```

Surprising sorts!

- < can also compare lists, so `.sort()` can sort a list of lists!

```
x = [[3], [2, 1]]
x[0] < x[1] # False
x.sort()
print(x) # [[2, 1], [3]]
```

Non-mutating sort

- `.sort()` mutates the list
- It's a method of lists: doesn't work on strings, ranges, tuples
- Also a built-in function to sort any sequence, by duplicating it into a list:

```
x = sorted("hello, world!")
print(x) # [" ", "!", ",", "d", "e",
         #  "h", "l", "l", "l", "o",
         #  "o", "r", "w"]
```

Least, greatest, gap

- Let’s solve exactly the question we started with: find the least, greatest, and greatest gap of a list

Least, greatest, gap

- Let’s solve exactly the question we started with: find the least, greatest, and greatest gap of a list

```
def lgg(lst: list[float]) -> tuple[float, float, float]:
    s = sorted(lst)
    greatestGap = 0.0
    for idx in range(0, len(lst)-1):
        gap = s[idx+1] - s[idx]
        if gap > greatestGap:
            greatestGap = gap
    return (s[0], s[-1], greatestGap)
```

Named parameters

CS114 M5

Read the documentation!

- Feeling a bit limited by `.sort()` and `sorted`? Remember that `help` can tell us how to use anything!

```
help([1, 2, 3].sort)
Help on built-in function sort:

sort(*, key=None, reverse=False) method of builtins.list
instance
    Sort the list in ascending order and return None.

    The sort is in-place (i.e. the list itself is modified) and stable
(i.e. the
    order of two equal elements is maintained).

    If a key function is given, apply it once to each list item and
sort them,
    ascending or descending, according to their function
values.

    The reverse flag can be set to sort in descending order.
```

```
Help on built-in function sort:

sort(*, key=None, reverse=False) method of builtins.list instance
    Sort the list in ascending order and return None.

    The sort is in-place (i.e. the list itself is modified) and stable (i.e. the
    order of two equal elements is maintained).

    If a key function is given, apply it once to each list item and sort them,
    ascending or descending, according to their function values.

    The reverse flag can be set to sort in descending order.
```

Help on built-in function sort:

sort(*, key=None, reverse=False) method of builtins.list instance
    Sort the list in ascending order and return None.

    The sort is in-place (i.e. the list itself is modified) and stable (i.e. the

An “instance” is one thing from a type of things.
`[1, 2, 3]` is an instance of a list,
1 is an instance of an int

    The reverse flag can be set to sort in descending order.

```
Help on built-in function sort:

sort(*, key=None, reverse=False) method of builtins.list instance
    Sort the list in ascending order and return None.

    The sort is in-place (i.e. the list itself is modified) and stable (i.e. the
```

The asterisk (somewhat confusingly) says that there are no normal parameters. We can't do `x.sort(42)`, because what would the 42 mean?

```
    The reverse flag can be set to sort in descending order.
```

```
Help on built-in function sort:

sort(*, key=None, reverse=False) method of builtins.list instance
    Sort the list in ascending order and return None.

    The sort is in-place (i.e. the list itself is modified) and stable (i.e. the
    [illegible]
```

But what are these things???

```
    If a key function is given, apply it once to each list item and sort them,
    ascending or descending, according to their function values.

    The reverse flag can be set to sort in descending order.
```

Reverse sort

- What if we want things backwards? `.sort` has a parameter for reversing, but it doesn't look like a normal parameter…
- It's a *named parameter* (or *keyword argument*). To pass it in, you have to name it:

Reverse sort

- What if we want things backwards? `.sort` has a parameter for reversing, but it doesn't look like a normal parameter…
- It's a *named parameter* (or *keyword argument*). To pass it in, you have to name it:

```
x = [2, 4, 6, 0, 1]
x.sort(reverse=True)
print(x) # [6, 4, 2, 1, 0]
```

```
Help on built-in function sort:

sort(*, key=None, reverse=False) method of builtins.list instance
    Sort the list in ascending order and return None.

    The sort is in-place (i.e. the list itself is modified) and stable (i.e. the
    order of two equal elements is maintained).

    If a key function is given, apply it once to each list item and sort them,
    ascending or descending, according to their function values.

    The reverse flag can be set to sort in descending order.
```

OK, we saw “reverse”, but what does this mean?

Key sort

- Sorting normally uses <
- This only works on things you can compare with <, but it's also pretty limited
- What if I wanted to sort strings by their *length*?
- `.sort()` provides a way to change the sorting criteria: a *key function*

Sort strings by length

```
x = ["an", "excellent", "list", "of", "strings"]
x.sort(key=len)
print(x) # ["an", "of", "list", "strings", "excellent"]
```

Sort ints even/odd

```
def parity(val: int) -> int:
    return val%2
lst = [8, 6, 7, 5, 3, 0, 9]
s = sorted(lst, key=parity)
print("After sorted, lst is", lst)
print("Sorted:", s)
lst.sort(key=parity)
print("After .sort, lst is", lst)

# [8, 6, 0, 7, 5, 3, 9]
```

Sort ints even/odd

```python
def parity(val: int) -> int:
    return val%2
lst = [8, 6, 7, 5, 3, 0, 9]
s = sorted(lst, key=parity)
```

NOTE: Sorting is “stable”. This means that if two distinct values could go in either order (such as 8 and 0 here) it won’t swap them.

```python
print("After sort, lst is", lst)

# [8, 6, 0, 7, 5, 3, 9]
```

In-lecture quiz (L11)

- https://student.cs.uwaterloo.ca/~cs114/quiz/
- Q1: What will this print?

```
def first(s: str) -> str:
    return s[0]
print(sorted(
    ["an", "aardvark", "ate", "ants"],
    key=first
))
```

A. Nothing or an error

B. `['an', 'aardvark', 'ate', 'ants']`

C. `['aardvark', 'an', 'ants', 'ate']`

D. `['ate', 'aardvark', 'an', 'ants']`

E. `['ate', 'ants', 'an', 'aardvark']`

In-lecture quiz (L11)

- https://student.cs.uwaterloo.ca/~cs114/quiz/
- Q2: What will this print?

```
def verst(s: str) -> str:
    return s[::-1]
print(sorted(
    ["an", "aardvark", "ate", "ants"],
    key=verst
))
```

A. Nothing or an error

B. `['an', 'aardvark', 'ate', 'ants']`

C. `['aardvark', 'an', 'ants', 'ate']`

D. `['ate', 'aardvark', 'an', 'ants']`

E. `['ate', 'ants', 'an', 'aardvark']`