Python

Python Comparisons

=	Assignment
==	Test of equality
!=	Test of inequality
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Python String Operations

`len(s)`	Get # characters in string
`s.lower()`	Convert to lowercase
`s.upper()`	Convert to uppercase
`s1 + s2`	Concatenate
`x in s`	Check if character is in string
`s.count(x)`	Count occurrences of character(s)
`s.startswith(x)`	Checks if string starts with character(s)
`s.endswith(x)`	Checks if string ends with character(s)
`s.split(x)`	Split string on character(s)
`x.join(list)`	Join list of strings on character(s)
`s.find(x, i)`	Returns position of character(s); (starts search at 'i', if specified)

Python Special String Characters

`\`	Escape character
`\t`	Tab
`\n`	New line

Python Printing and Formatting

We've explored four approaches to printing:

`print("text", s)`	Separate items with commas
`print("text " + s)`	Concatenate items directly
`print(f"text {s}")`	Use implicit formatting
`print("text {s}".format(s='s'))`	Use explicit formatting

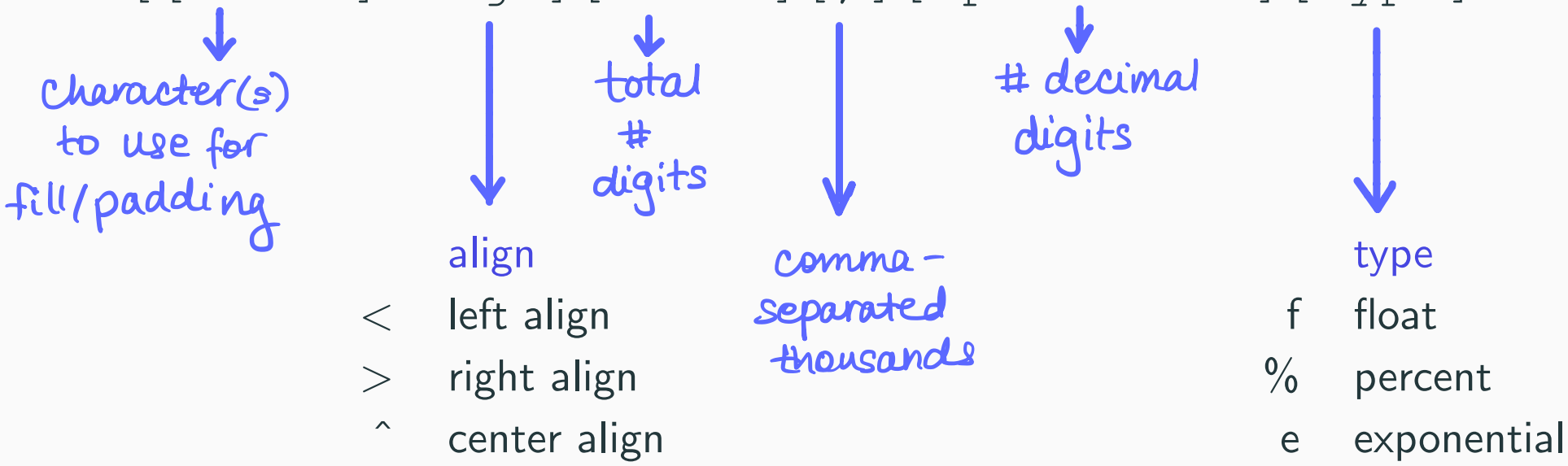
Python String Formatting

We format with syntax like:

```
{field_name:format_spec}
```

where `format_spec` takes the following form:

```
[[fill ]align][width][,][.precision][type]
```



Python String Formatting

As an example, if we type

```
{x : y^z.wf}
```

We will format x to be a center-aligned (^) float (f) with w decimal digits of precision. The printed string will have z digits total, and any extra space will be filled with y.

Python String Indexing and Slicing

`s[i]`	Return character at position i
`s[-i]`	Return character at position i from the right
`s[i:]`	Return characters from i (inc) onwards
`s[:j]`	Return characters up to j (exc)
`s[i:j]`	Return characters from i (inc) to j (exc)

inc = inclusive, *exc* = exclusive

Recall:

- String indexing starts at position 0.
- We can use the same indexing and slicing approach with lists/tuples.

Python Strings

Some other things to remember about strings:

- Strings are case sensitive
- We can create a multi-line string with blockquotes, e.g. : `"""string"""` or `'''string'''`
- We can use $>$ or $<$ to do string comparisons; order is determined by dictionary order, where:

  numbers $<$ uppercase letters $<$ lowercase letters

Python Boolean Variables

Recall the rules for Boolean combinations:

```
True and True    =  True
True and False   =  False
False and False  =  False

True or True     =  True
True or False    =  True
False or False   =  False

not True         =  False
not False        =  True
```

Python if-else Statements

`if cond:`	Execute if condition is True
`elif cond:`	Execute if condition is True, and no preceding 'if' statement was executed
`else:`	Execute if no preceding 'if' statement was executed ("catch-all")

Recall:

- Indentation determines the lines affected by the 'if' statement
- We can nest 'if' statements

Python List Functions

`len(list)`	Get # elements in list
`sorted(list)`	Return sorted list
`max(list)`	Return maximum element
`min(list)`	Return minimum element
`sum(list)`	Sum all (numeric) list elements
`list.pop(i)`	Remove item at specified position**
`list.insert(i,x)`	Insert item at specified position**
`list.append(x)`	Append an item to list**
`list1.extend(list2)`	Append another list**
`list1 + list2`	Add two lists together
`x in list`	Check if item is in list
`list.index(x)`	Get the index of item
`list.count(x)`	Count appearances of item in list

** Modifies in place.