Syntax

- Indentation matters
- Variables
  - No type declaration (python thinks its smart)
  - `x = 1`
  - `y = "4"`
- Casting
  - `int("3")`
  - `str(5)`
  - `float("5")`
  - `type(x)` returns the type of the variable `x`
- Comments
  - `# single line`

  -
    ```
    ‘‘‘hacky but acceptable way
    to use multi-line comments’’’
    ```
- Variable name errors
  - `8var`
  - `My+var`
  - `My var`
- Acceptable: `myVar`, `_my_var`, `MYVAR`
- Standards for multi-word variables in order of observed popularity
  - `snake_case`
  - `camelCase`
  - `PascalCase`
- Assigning multiple values
  - `x, y, z = "hello", 6, 8.0`
  - `x = y = z = "hello"`
  - `x, y, z = ["hello", 5, 8.0]`
- Type of variables matters with operations
  - `5 + 6`
  - `"he" + "llo"`
  - `5 + "llo"` This does not work, it results in an error.

Data Types

- Text Type: str
- Numeric Types: int, float, complex
- Sequence Types: list, tuple, range
- Mapping Type: dict
- Set Types: set, frozenset
- Boolean Type: bool
- Binary Types: bytes, bytearray, memoryview

Strings

- Strings are kind of treated as a list of characters (there is not character type)
  - `"hello"[1]` returns `"e"`
- Can be defined using `"hello"` or `'hello'`
- Concatenate strings `"he" + "llo"`

Booleans

- `True` / `False`
- (everything else / 0)
- Case matters, `True`, not `true`
- `5 < 8`
- `"hi" == "hello"`

Casting

- Often, numerical values in a file will be interpreted as a string
- We cant add `"5" + 6` to get 11
- We need to tell python that we want it to be treated as a numerical value (int, float, etc.)
- To do this, we cast the item to a new type
  - `int("5") = 5`
  - `int("5") + int("6") = 11`
  - `str(74) = "74"`

Operators

Arithmetic Operators

`+`	Addition	`x + y`
`-`	Subtraction	`x - y`
`*`	Multiplication	`x * y`
`/`	Division	`x / y`
`%`	Modulus	`x % y`
`**`	Exponentiation	`x ** y`
`//`	Floor Division	`x // y`

Assignment Operators

`=`	`x = 3`	`x = 3`
`+=`	`x += 3`	`x = x + 3`
`-=`	`x -= 3`	`x = x - 3`
`*=`	`x *= 3`	`x = x * 3`
`/=`	`x /= 3`	`x = x / 3`
`%=`	`x %= 3`	`x = x % 3`
`//=`	`x //= 3`	`x = x // 3`
`**=`	`x **= 3`	`x = x ** 3`
`&=`	`x &= 3`	`x = x & 3`
`|=`	`x |= 3`	`x = x | 3`
`^=`	`x ^= 3`	`x = x ^ 3`
`>>=`	`x >>= 3`	`x = x >> 3`
`<<=`	`x <<= 3`	`x = x << 3`

Logical

`and`	Returns `True` if both statements are true	`x < 5 and x < 10`
`or`	Returns `True` if one of the statements is true	`x < 5 or x < 4`
`not`	Reverse the result, returns `False` if the result is `True`	`not (x < 5 and x < 10)`

Identity

`is`	Returns `True` if both variables are the same object	`x is y`
`is not`	Returns `True` if both variables are not the same object	`x is not y`

Membership

`in`	Returns `True` if a sequence with the specified value is present in the object	`x in y`
`not in`	Returns `True` if a sequence with the specified value is not present in the object	`x not in y`

Lists

- `myList = [5, "a", "list", 5, 6]`
- Index starts are 0

- Lists are mutable
  - `myList[2] = "change the second indexed value"`
- `len(myList) -> 5`
- Lists are one of four collection types:
  - Lists
  - Tuples
  - Sets
  - Dictionaries

List Items

- `myList[2] -> "list"`
- `myList[-1] -> 6`
- `myList[1:4] -> ["a", "list", 5] # end is exclusive`
- `myList[:4] -> [5, "a", "list", 5]`
- `myList[3:] -> [5, 6]`
- `"a" in myList -> True`

Modify Lists

- `myList[1] = "B" -> [5, "B", "list", 5, 6]`
- `myList[1:3] = ["C", "D"] -> [5, "C", "D", 5, 6]`
- `myList[1:2] = ["C", "D"] -> [5, "C", "D", "list", 5, 6]`
- `myList[1] = ["C", "D"] -> [5, ["C", "D"], "list", 5, 6]`