Key Term

List comprehension: A compact way to process and generate lists in Python.

```
# Double each number in a list
numbers = [1, 2, 3, 4]
doubled = [x*2 for x in numbers]
print(doubled) # [2, 4, 6, 8]
```

```
[2, 4, 6, 8]
```

Lambda function: An anonymous inline function defined without a name.

```
# Define a lambda function to cube numbers
cube = lambda x: x**3
print(cube(3)) # 27
```

```
27
```

Numpy array: A fast numeric array structure in Python provided by Numpy.

```
import numpy as np

# Create a Numpy array
arr = np.array([1, 2, 3])
print(arr) # [1 2 3]
```

```
[1 2 3]
```

Pandas DataFrame: A tabular data structure provided by Pandas for data analysis.

```
import pandas as pd

# Create a Pandas DataFrame
data = {'Name': ['Alice', 'Bob'], 'Age': [25, 30]}
df = pd.DataFrame(data)
print(df)
```

```
   Age   Name
0   25  Alice
1   30    Bob
```

Matplotlib: A Python plotting library for data visualization.

```
import matplotlib.pyplot as plt

# Simple Matplotlib line plot
x = [1, 2, 3, 4]
y = [2, 4, 6, 8]
plt.plot(x, y)
plt.show()
```

Sequences in Python

Data Structures

Organized approaches to storing data and working with data

Data structures in Python are

Sequences

Ordered finite collections of data

Sequence Types

- Lists — `[ 1, 2, 3 ]`
- Tuples — `( 1, 2, 3 )`
- Strings — `"123"`
- Binary Strings — `b"123"`
- Range Objects — `range(1, 4)`

- All of the sequence data type share some operations

These can be grouped into

- membership operations
- Indexing operations
- Slicing operations
- Interrogation operations
- Math operations

membership: • used in and not in to test the value is in the list (be a member?)

Membership

sequence is of type list

```
[114]: 3 in [ 1, 2, 3, 4, 5 ]
[114]: True
```

Membership

```
[115]: 12 in [ 1, 2, 3, 4, 5 ]
[115]: False
```

Membership

```
[116]: 12 not in [ 1, 2, 3, 4, 5 ]
[116]: True
```

Indexing: use to access a particular item in a sequence

Indexing

This sequence is of type string

```
[118]: name = "Monteverdi"
[119]: name[0]
[119]: 'M'
[121]: name[4]
[121]: 'e'
```

index to access string position zero or first position

```
[122]: name[-1]
[122]: 'i'
[123]: name[-2]
[123]: 'd'
```

last position

Slicing:

Slicing

```
[126]: name = "Palestrina"
[127]: name[2:5]
[127]: 'les'
[128]: name[:5]
[128]: 'Pales'
[129]: name[4:]
[129]: 'strina'
[130]: name[-3:]
[130]: 'ina'
[131]: name[:]
[131]: 'Palestrina'
[132]: name[::2]
[132]: 'Plsrn'
[133]: name[::-2]
[133]: 'aitea'
```

start with index=2 — end with index = 4 (always n-1)

= default = zero

to the last index in sequence

get all index

start with index=0 — step of 2 until end

start at the end to first index

Interrogation:

Interrogation

```
[134]: len(name)
[134]: 10
[135]: min(name)
[135]: 'P'
[136]: max(name)
[136]: 't'
[137]: max(['Free', 2, 'b'])
TypeError                                 Traceback (most recent call last)
/var/folders/29/5dyw2t0n71v13sp1980c30g80000gr/T/ipykernel_35045/3929585417.py in <module>
----> 1 max(['Free', 2, 'b'])
TypeError: '>' not supported between instances of 'int' and 'str'
[138]: name.count('e')
[138]: 1
[139]: name.index('i')
[139]: 7
```

→ length of the sequence

→ minimum item in sequence (ASCII)

↗ so Cap. letters are lower than lower letters.

sequences can contain different types of items

↳ than they cannot use interrogation operations

↳ count the no. of times an items shows up in a sequence

↳ find where the first index of the item is placed

Math:

Math

connect together

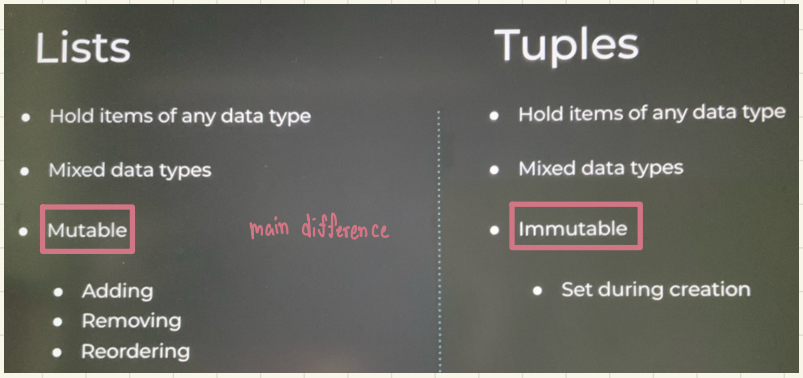
```
[140]: "prefix" + "-" + "postfix"
[140]: 'prefix-postfix'
[141]: [1,2] + [3,4]
[141]: [1, 2, 3, 4]
[142]: [0,2] * 4
[142]: [0, 2, 0, 2, 0, 2, 0, 2]
```

→ extend the list

↗ times

Lists and Tuples in Python

- They are closely related



Creation

```
[143]: tuple()
[143]: ()
[144]: list()
[144]: []
[145]: tup = (1,2)
       tup
[145]: (1, 2)
[146]: some_list = [1,2,3]
       some_list
[146]: [1, 2, 3]
[147]: some_tuple = (1,)
       some_tuple
[147]: (1,)
[148]: some_tuple = (1)
       some_tuple
[148]: 1
[149]: tup = 1,
       tup
[149]: (1,)
```

- create one item in tuple, still has comma
- in sequence, it will be interpreted as just a grouping
- take the comma out ← the variable just set to integer 1
- can also create a tuple just by having the items with a common trailing (no need parenthesis)

```
[150]: name = "Beethoven"
       letters = list(name)
       letters
[150]: ['B', 'e', 'e', 't', 'h', 'o', 'v', 'e', 'n']
[151]: tuple(name)
[151]: ('B', 'e', 'e', 't', 'h', 'o', 'v', 'e', 'n')
```

← string type
We can change it to another type of sequence by doing "casting"

Adding and Removing Items

```
[152]: composers = ['Vivaldi', 'Mozart']
       composers
[152]: ['Vivaldi', 'Mozart']
[153]: composers.append('Brahms')
       composers
[153]: ['Vivaldi', 'Mozart', 'Brahms']
[154]: composers.insert(0,"Pérotin")
       composers
[154]: ['Pérotin', 'Vivaldi', 'Mozart', 'Brahms']
[156]: composers.pop(0)
[156]: 'Pérotin'
```

item → add item to a list

→ to specify the index and add item to

if we don't specify the index, it will remove the last index by default

→ to remove an item from a list

```
[157]: vips = [ 'Da Vinci', 'Franklin']
       vips
[157]: ['Da Vinci', 'Franklin']
[158]: vips.extend(composers)
       vips
[158]: ['Da Vinci', 'Franklin', 'Vivaldi', 'Mozart']
[159]: lists = [[]] * 4
       lists
[159]: [[], [], [], []]
[160]: lists[-1].append(4)
       lists
[160]: [[4], [4], [4], [4]]
```

main list → desire list

to add list together

list as the item

we can have a list of lists

it's the same instance of the same list

appear to be different lists

← it's tricky

list comprehension ← recommended rather than by multiplication to avoid that problem

```
[161]: lists = [[] for _ in range(4)]
       lists[-1].append(4)
       lists
[161]: [[], [], [], [4]]
```

Ordering

```
[162]: name = "Beethoven"
[163]: letters = list(name)
       letters
[163]: ['B', 'e', 'e', 't', 'h', 'o', 'v', 'e', 'n']
[164]: letters.sort()
       letters
[164]: ['B', 'e', 'e', 'e', 'h', 'n', 'o', 't', 'v']
[165]: letters.reverse()
       letters
[165]: ['v', 't', 'o', 'n', 'h', 'e', 'e', 'e', 'B']
```

ASC ↑ lowest

DESC ↑

Strings in Python

Strings

Strings → one of the type in sequence

```
[166]: 'Here is a string'
```

```
[166]: 'Here is a string'
```
→ Represented by any unicode characters enwrapped in quotes

```
[167]: "Here is a string" == 'Here is a string'
[167]: True
```
double and single quotes are the same. it's considered their content

```
[168]: 'Here "is" a string'
[168]: 'Here "is" a string'
```

multiline string ↓ by triple quotes

```
[ ]: a_very_large_phrase = """
Wikipedia is hosted by the Wikimedia Foundation,
a non-profit organization that also hosts a range of other projects.
"""

print(a_very_large_phrase)
```

```
[170]: print("Ludwig\tBeethoven")
Ludwig  Beethoven
```
tap

```
[171]: print("Ludwig\nBeethoven")
Ludwig
Beethoven
```
new line

```
[172]: windows_path = "c:\tam\Projects\now"
print(windows_path)
c:      am\Projects
ow
```
cause an error

→ to make it a raw string

```
[173]: windows_path = r"c:\tam\Projects\now"
print(windows_path)
c:\tam\Projects\now
```

Raw String all characters are interpreted purely as what they are

```
[174]: bill = "William Byrd"
bill
[174]: 'William Byrd'
```
→ both capitalization

```
[175]: bill.capitalize()
[175]: 'William byrd'
```
↳ make it one cap. in first letter

```
[176]: bill.lower()
[176]: 'william byrd'
```
lower case

```
[177]: bill.upper()
[177]: 'WILLIAM BYRD'
```
upper case

Interrogation

Interrogation

```
[184]: leo = 'Léonin'
leo
[184]: 'Léonin'
```

```
[186]: leo.index('Lé')
[186]: 0
```
return the index of a substring

```
[187]: leo.index('r')
```
→ it does not exist

```
ValueError                                Traceback (most recent call last)
/var/folders/29/5dyw2t0n71v13sp1980c30g80000gr/T/ipykernel_35045/2596832993.py in <module>
----> 1 leo.index('r')

ValueError: substring not found
```

```
[188]: leo.find('r')
[188]: -1
```
→ mean the item is not in the string

```
[189]: leo.startswith('L')
[189]: True
```
→ check the first string

```
[190]: leo.endswith('in')
[190]: True
```

Content

Content Type

```
[191]: 'abc123'.isalpha()
[191]: False
```
↳ is it alphabetic

```
[192]: 'abc123'.isalnum()
[192]: True
```
↳ is it only alphabetic and numeric contents

```
[193]: 'lowercase'.islower()
[193]: True
```

```
[194]: 'lowercase'.isupper()
[194]: False
```

```
[195]: one = '1'
[196]: one.isnumeric()
[196]: True
```
useful when we want to turn string → numerical

Format Strings

Format strings

- Introduced in Python 3.6.
- Prefixed by either a 'F' or 'f' (doesn't matter)
- Values inserted at runtime using replacement fields.

```
[197]: f"Example {1}"
[197]: 'Example 1'
```
→ curly brackets = replacement field

Insert variable into replacement field

```
[198]: strings_count = 5
frets_count = 24
f"Noam Pikelny's banjo has {strings_count} strings and {frets_count} frets"
[198]: "Noam Pikelny's banjo has 5 strings and 24 frets"
```

Insert expression into replacement field

```
[199]: a = 12
b = 32
f"{a} times {b} equals {a+b}"
[199]: '12 times 32 equals 384'
```
with expression

Index list in string replacement fields

```
[200]: players = ["Tony Trischka", "Bill Evans", "Alan Munde"]

f"Performances will be held by {players[1]}, {players[0]}, and {players[2]}"
[200]: 'Performances will be held by Bill Evans, Tony Trischka, and Alan Munde'
```

Padding a number . to do a formatting of the item inserted

```
[201]: lucky_num = 13
f"To pad the number {lucky_num} to 5 places:{lucky_num:5d}"
[201]: 'To pad the number 13 to 5 places:    13'
```
pad with white space

formatting generally happens after a colon

Setting padding value at runtime

```
[202]: luckey_num = 13
padding = 5
f"To pad the number {lucky_num} to {padding} places:{lucky_num:{padding}d}"
[202]: 'To pad the number 13 to 5 places:    13'
```
nested replace

Commas

```
[203]: balance = 1232980098
print(f"Your balance is {balance:,}")
Your balance is 1,232,980,098
```
wow!

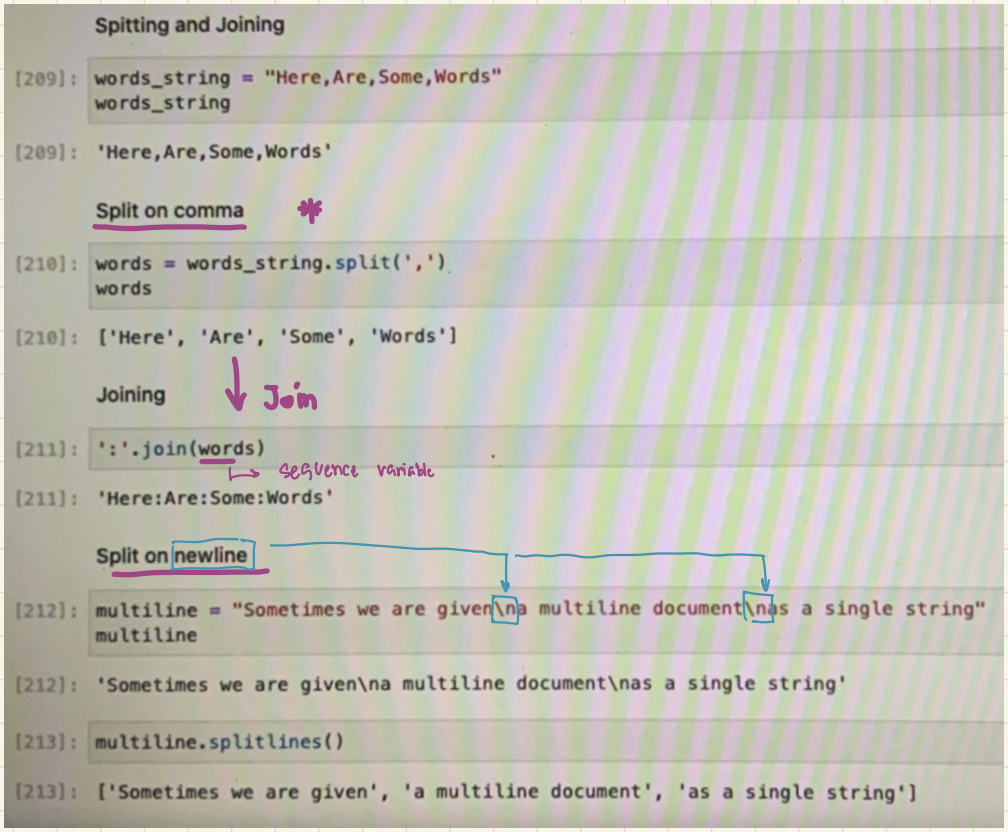
Continue Strings in Python

Manipulate strings

Removing Whitespace

```
[204]: ship = " The Yankee Clipper "
       ship

[204]: ' The Yankee Clipper '

[205]: ship.strip()

[205]: 'The Yankee Clipper'

[206]: ship.lstrip()

[206]: 'The Yankee Clipper '

[207]: ship.rstrip()

[207]: ' The Yankee Clipper'
```

→ whitespace that's leading or trailing is removed



Spitting and Joining

```
[209]: words_string = "Here,Are,Some,Words"
       words_string

[209]: 'Here,Are,Some,Words'
```

Split on comma

```
[210]: words = words_string.split(',')
       words

[210]: ['Here', 'Are', 'Some', 'Words']
```

Joining → Join

```
[211]: ':'.join(words)

[211]: 'Here:Are:Some:Words'
```

→ sequence variable

Split on newline

```
[212]: multiline = "Sometimes we are given\na multiline document\nas a single string"
       multiline

[212]: 'Sometimes we are given\na multiline document\nas a single string'

[213]: multiline.splitlines()

[213]: ['Sometimes we are given', 'a multiline document', 'as a single string']
```

Creating Range Objects in Python

Range objects

can specify to start number

```
[215]: range(2,10)

[215]: range(2, 10)

[216]: list(range(1, 10))

[216]: [1, 2, 3, 4, 5, 6, 7, 8, 9]

[217]: import sys

       r = range(3)
       sys.getsizeof(r)

[217]: 48

[218]: r = range(10000000)
       sys.getsizeof(r)

[218]: 48
```

get the size of an object in Python

same size, so you can taking long ecum numbers without taking up a lot of memory

```
[214]: range(10)

[214]: range(0, 10)
```

not including list

```
l = list(range(10000000))
sys.getsizeof(l)

80000056
```

start, end, steps

```
[220]: list(range(0,10,2))

[220]: [0, 2, 4, 6, 8]
```

start, end, reverse step

```
[221]: list(range(10, 0, -2))

[221]: [10, 8, 6, 4, 2]
```

most common to use for range object

```
[222]: start = 0
       for count in range(5):
           start += count
           print(start)

       0
       1
       3
       6
       10
```