Structured Types

Tuples

- storage containers
- can contain mixed values
- immutable
- addition creates a new tuple

```python
# these are tuples
new_pair = (3.2, 4)
new_quadruple = ("Story number", 3, "is", True)
```

```python
# these are tuples
tuple_empty = ()
tuple_str = ("Story",)
tuple_int = (3,)
tuple_float = (3.14159,)
```

```python
# these are NOT tuples
example_int = (100)
example_float = (100.001)
```

Adding Tuples Creates Entirely New Tuple

```
pair = (3.2, 4)
quadruple = ("Story number", 3, "is", True)
new_tuple = pair + quadruple
print(new_tuple)
```

```
(3.2, 4, 'Story number', 3, 'is', True)
```

Accessing Tuple Elements with [ ]

```
a = (1, 2, 3)
print(a[0])
```

what prints, what is type(a)?

```
a = (1, 2, 3)[0]
```

what is type(a)?

Tuples are Immutable

```
a = (1, 2, 3)
a[0] = 10
```

what is type(a)?
what is type(a[0])?
what is this code doing?

CRASH

Lists

- storage containers
- can contain mixed values
- mutable
- addition creates a new list
- **appending** alters an existing list

```
# these are lists
new_pair = [3.2, 4]
new_quadruple = ["Story number", 3, "is", True]
```

```
# these are lists
list_empty = []
list_str = ["Story"]
list_int = [3]
list_float = [3.14159]
```

Adding Lists Creates Entirely New List

```
pair = [3.2, 4]
quadruple = ["Story number", 3, "is", True]
new_list = pair + quadruple
print(new_list)
```

```
[3.2, 4, 'Story number', 3, 'is', True]
```

Accessing List Elements with [ ]

```
a = [1, 2, 3]
print(a[0])
```

what prints, what is type(a)?

```
a = [1, 2, 3][0]
```

what is type(a)?

List are Mmutable

```
a = [1, 2, 3]
a[0] = 10
```

what is type(a)?
what is type(a[0])?
what is this code doing?

RUNS

Appending to an Existing List

```
a = [1, 2, 3]
a.append(1)
print(a)
```

"dot" notation, for convenience

```
[1, 2, 3, 1]
```

```
a = [1, 2, 3]
list.append(a, 1)
print(a)
```

```
[1, 2, 3, 1]
```

Appending vs. Adding

- Another way to say: changing existing list vs. making a new list
- ^^^Not a consideration for tuples, tuples must always be made anew

Making New

- new memory chunk needed
- every element has to be copied

Changing Existing (appending)

- new element is simply added onto the end of items in the memory chunk containing the list
- (rarely, a larger memory chunk will have to allocated, and all items copied)

Tuple vs List

If you are frequently updating a container…which one would be faster?

Why?

LIST, less copying

Slicing

applies to any type that is indexable

- lists, tuples, strings

indices given with integer index!

```
a = [0, 'elem1', 'elem2', 3, 'this is the fourth element', '5', '6.0000', 7, 8.0000]
# retrieve elements 0 up to 3rd (not including 3rd)
print(a[0:3])
```

```
[0, 'elem1', 'elem2']
```

```
a = [0, 'elem1', 'elem2', 3, 'this is the fourth element', '5', '6.0000', 7, 8.0000]
# retrieve elements 0 up to 8th (not including 8th), in step size of 2
print(a[0:8:2])
```

```
[0, 'elem2', 'this is the fourth element', '6.0000']
```

Slicing to the end

```
a = [0, 'elem1', 'elem2', 3, 'this is the fourth element', '5', '6.0000', 7, 8.0000]
# retrieve elements 5 up to 8th (including 8th)
print(a[5:])
```

```
['5', '6.0000', 7, 8.0]
```

Slicing backward

```
a = [0, 'elem1', 'elem2', 3, 'this is the fourth element', '5', '6.0000', 7, 8.0000]
# retrieve elements from end to 0 (not including 0) backward (step size -1)
print(a[:0:-1])
```

```
[8.0, 7, '6.0000', '5', 'this is the fourth element', 3, 'elem2', 'elem1']
```

```
a = [0, 'elem1', 'elem2', 3, 'this is the fourth element', '5', '6.0000', 7, 8.0000]
# retrieve elements from end to 0 (including 0) backward (step size -1)
print(a[::-1])
```

```
[8.0, 7, '6.0000', '5', 'this is the fourth element', 3, 'elem2', 'elem1', 0]
```

Slicing with reference to the end

```
a = [0, 'elem1', 'elem2', 3, 'this is the fourth element', '5', '6.0000', 7, 8.0000]
# retrieve elements 2 up to 2nd to last (not including 2nd to last)
print(a[2:-2])
```

```
['elem2', 3, 'this is the fourth element', '5', '6.0000']
```

Google Form

What will this print?

```
a = [0, 'elem1', 'elem2', 3, 'this is the fourth element', '5', '6.0000', 7, 8.0000]
# retrieve a string and do further indexing
print(a[4][8:-8])
```

https://forms.gle/NTjob5R2We6PiQ3a6

pretty_print_list

def pretty_print_list(values: Iterable[int]) -> str:

"""Pretty print a list without brackets and adding commas."""

```
from typing import Iterable

def prettyprint(values: Iterable[int]) -> str:
  return str(values)[1:-1]
```

Iterating through list or tuple or string or range

```
a = [0, 'elem1', 'elem2', 3, 'this is the fourth element', '5', '6.0000', 7, 8.0000]

for item in a:
  print(item)
```

Explore Tuples and Lists

Tuples:
https://github.com/allegheny-college-cmpsc-101-spring-2024/course-materials/blob/main/notes/20240221_structured_types_tuples.ipynb

Lists (and slicing):
https://github.com/allegheny-college-cmpsc-101-spring-2024/course-materials/blob/main/notes/20240222_structured_types_lists.ipynb

Assignment on Structured Types (0/100): https://classroom.github.com/a/Fxj3JGsj

Friday

- Aliasing (referring to the same object with more than one name)
- List comprehensions
- Sets
- Dictionaries
- Assignment on Intersection Algorithms (graded)