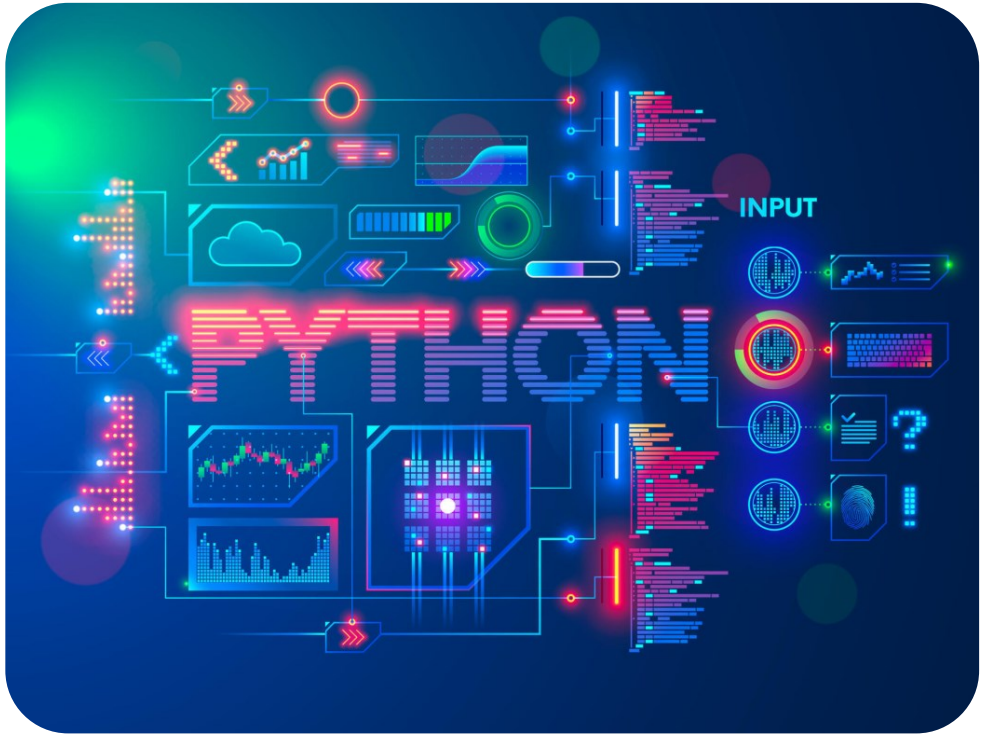
COLUMBUS STATE
COMMUNITY COLLEGE



WIIT 7740: Scripting with Python

Week 4: Lists and Tuples

Creating a List

```
items = list()
# OR
items = []
```

Create a List

- Initialize as empty

```
items = list()
items = []
```

- Initialize with one element

```
items = ["Alpha"]
```

- Initialize with many elements

```
items = ["Alpha", "Bravo", "Charlie"]
```

Accessing Data in a List

- Initialize
  - `items = ["Alpha", "Bravo", "Charlie"]`

- Access item at index [x]
  - `items[0]` → `"Alpha"`
  - `items[1]` → `"Bravo"`
  - `items[2]` → `"Charlie"`

- How many elements are in the list?
  - `len(items)` → `3`

Contents of lists

- List of integers
  - `[56, 17, 92, 45]`
- List of strings
  - `["Papa", "Yankee", "Tango", "Hotel", "Oscar", "November"]`
- List of mixed data
  - `[6, "feet", 9.5, "inches"]`
- List of lists
  - `[ ["Chicago", "IL"], ["Atlanta", "GA"], ["Cincinnati", "OH"] ]`

Nested Lists

- Initialize
  - ```
    places = [
          ["Chicago", "IL"],
          ["Atlanta", "GA"],
          ["Cincinnati", "OH"]
    ]
    ```
- Access element by using index [x][y]
  - `places[0][0]` → `"Chicago"`
  - `places[0][1]` → `"IL"`
  - `places[2][1]` → `"OH"`
    ```

Joining Lists

- Operator **+** joins lists together
  - `intro = ['Foreword', 'Prologue']`
  - `main  = ['Chapter 1', 'Chapter 2', 'Chapter 3']`
  - `outro = ['Epilogue', 'Appendix']`
  - `sections = intro + main + outro`

  - `sections == ['Foreword', 'Prologue', 'Chapter 1', 'Chapter 2', 'Chapter 3', 'Epilogue', 'Appendix']`

- Preserves existing lists

Mutating a List

- Initialize
  - `items = ["Alpha", "Bravo", "Charlie"]`
- Mutate
  - `items[2] = "Golf"`
- Side effect
  - `items → ["Alpha", "Bravo", "Golf"]`

Mutating a List by Adding Elements

- Appending
  - `items = [“a”, “b”, ”c”]`
  - `items.append(“d”)`

- Extending
  - `items.extend( [“e”, “f”, ”g”] )`

Mutating a List by Removing Elements

- Deleting an element at an index
  - `del items[0]`
- Popping out the last element
  - `last_item = items.pop()`
- Popping out element at index
  - `some_item = items.pop(n)`
- Removing element by value
  - `items.remove("Golf")`

Comparison

- What is the difference?
  - `item = items[n]`
  - `item = items.pop(n)`

Sorting a List

- Initialization
  - `items = [77, 13, 91, 28]`

- What is the difference?

  - `sorted_items = sorted(items)`

  - `items.sort()`

Another Comparison

- What is the difference?
  - `C = A + B`
  - `A.extend(B)`

Splitting and Joining

- Split
  - `phrase = "x,y,z"`
  - `phrase.split(",") → ["x", "y", "z"]`
- Join
  - `items = ["x", "y", "z"]`
  - `delimiter = ":"`
  - `delimiter.join(items) → "x:y:z"`

List Slices

- Access slices of list

```
items = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y',
'z']

items[11:15] → ? # Elements 11 - 14 as a new list

items[ starting_from : up_to_but_excluding ]
```

- Mutating slices of list

```
items[25:25] = ['and']

items[11:15] = 'lmno'
```

Tuples: What are they?

- One of the built-in data types used to store collections of data
- Tuples are used to store multiple items as a single variable
- They are immutable, meaning, that elements cannot be added/removed/modified after it has been created
- Indexed like a List or a String

Tuples: Examples

- Ordered sequence of any values

- `(latitude, longitude)`
- `(first_name, middle_name, last_name)`
- `(5, "feet", 2, "inches")`
- `(singleton,)`
- `()`

Accessing Values from Tuples

```
t = ('alpha', 'bravo', 'charlie')

t[0]     →   "alpha"

t[1]     →   "bravo"

t[2]     →   "charlie"
t[-1]    →   "charlie"
```

Tuples are immutable

Tuples (like strings) cannot be mutated after they are created

```
t = ('alpha', 'bravo', 'charlie')
```

```
t[0] = 'delta'
```

Will raise exception!

```
TypeError: 'tuple' object does not support item assignment
```

Tuples have Length

```
t = ('alpha', 'bravo', 'charlie')

len(t)   →   3
```

Tuples can be Joined Together

```
x = ('April', 28)
y = (5, 38, 'PM')

x + y    →    ('April', 28, 5, 38, 'PM')
```

Set multiple variables from a tuple

```
measurement = (26.2, 'miles')

(quantity, unit) = measurement
```

```
quantity    →   26.2
unit        →   'miles'
```

Swap variable values

```
(yours, mine) = (mine, yours)
```

Otherwise, you would have to do it this way:

```
temp = yours
yours = mine
mine = temp
```

Check tuples for equality

```
('April', 24) == ('April', 24)      → True

('April', 24) == ('April', 25)      → False

('April', 24) == ('May', 24)        → False

('April', 25) == ('April', 25, 2021) → False
```

Compare tuples

- Can compare tuples with any inequality operator
  - `<` `<=` `>` `>=`
- Compares values at respective positions

```
(2021, 4, 30) < (2021, 5, 1)          → True
```

1. Compare the n-th items of both tuples (starting with the zero-th index) using the `==` operator. If both are equal, repeat this step with the next item.
2. For two unequal items, the item that is “less than” makes the tuple, that contains it, also “less than” the other tuple.
3. If all items are equal, both tuples are equal.
4. If one tuple runs out of items during step 1, the shorter tuple is “less than” the longer one.

Quick References: Tuples

- Tuples are essentially immutable lists:

`tuple() → ()`	`(0, 1, 200) < (0, 2, 3)`	`(a,)`
`len(x)`	`x, y = y, x`	`(a, b) + (c, d)`

Glossary: Keywords & Terms

Keywords

`return`
send a Python object from inside the function back to the caller code; the function immediately terminates

Example:

```
my_list = list()
```

the `list()` function does some work and "returns" a list object to be stored in `my_list`

`yield`
send a Python object from inside the function back to the caller code, while maintaining enough context to return to the function

```
def item_generator():
    yield 5
    yield 'item'
    yield True

my_items = item_generator()
for i in my_items:
    print(i)
```

Output:

```
5
'item'
True
```

Terms

initialize

create a new object. Example: `my_list = []`

(initializes a new list)

mutate

modify an object. Objects that are considered "immutable" cannot be mutated.

side effects

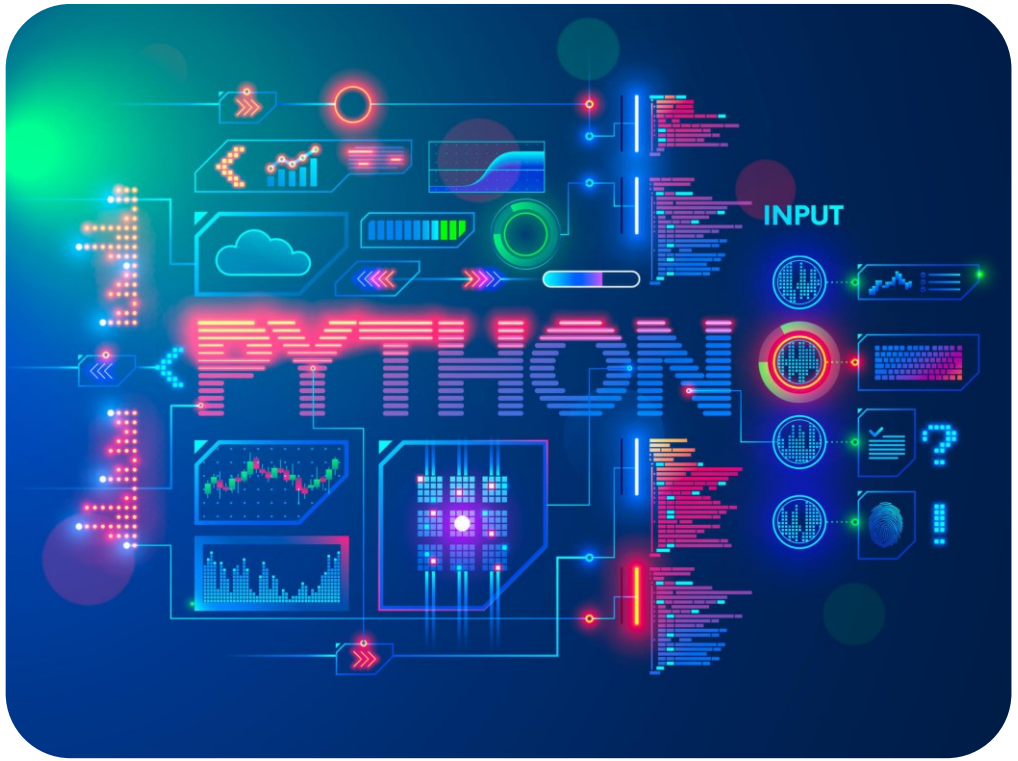
unintended processes or changes that occur when a program or portion of code is ran.

Glossary: Methods

Methods

- **`list()`**
  - constructor that returns a list object in python
  - alternatively `[]` can be used to initialize a list
- **`string.split(separator, maxsplit)`**
  - breaks a string into multiple parts specified by a separator
- **`string.join(iterable)`**
  - joins the **`string`** delimiter between all elements in an iterable (list, tuple, etc.) returning a new string
- **`list.sort(reverse, key)`**
  - sorts the items of a list in ascending or descending order
  - the key argument is used for the sort comparison if needed
- **`list.remove(element)`**
  - removes the first matching element in a list, does not return anything
- **`list.pop(index)`**
  - removes the element at the given index and returns it
- **`list.extend(iterable)`**
  - adds all elements of an iterable (list, tuple, etc.) to the end of a list
- **`list.append()`**
  - adds an element to the end of the list

COLUMBUS STATE
COMMUNITY COLLEGE



Practice Coding:
Class Activity