FinTech @ IU Python Session #3 – Data Structures and Looping

Prepared by

Gabriel Shores
Director of Technology – FinTech @ IU
https://www.linkedin.com/in/gabriel-shores-379b81291/

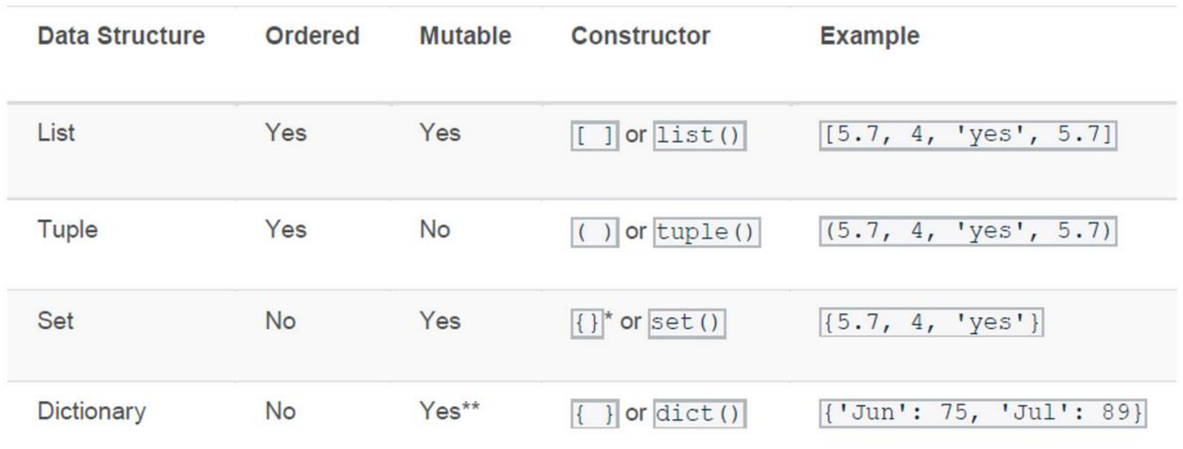
10/18/2024

Why Are Data Structures Important

- Allows us to store, access, and change data efficiently

- Makes programs more scalable

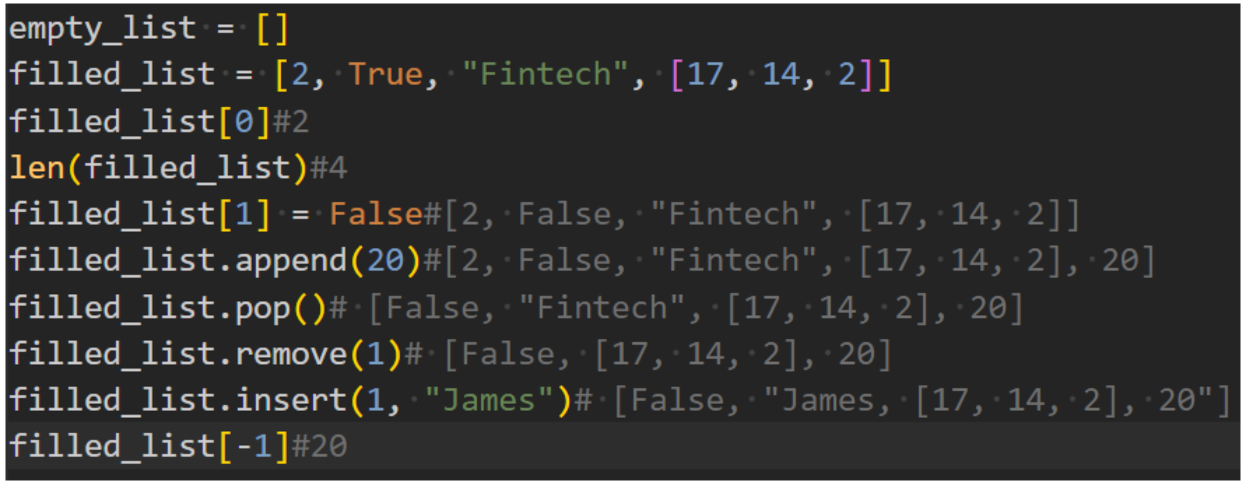
- Essential for Computer Science and Data Science algorithmic design



Data Structure	Ordered	Mutable	Constructor	Example
List	Yes	Yes	`[ ]` or `list()`	`[5.7, 4, 'yes', 5.7]`
Tuple	Yes	No	`( )` or `tuple()`	`(5.7, 4, 'yes', 5.7)`
Set	No	Yes	`{}`* or `set()`	`{5.7, 4, 'yes'}`
Dictionary	No	Yes**	`{ }` or `dict()`	`{'Jun': 75, 'Jul': 89}`

Lists

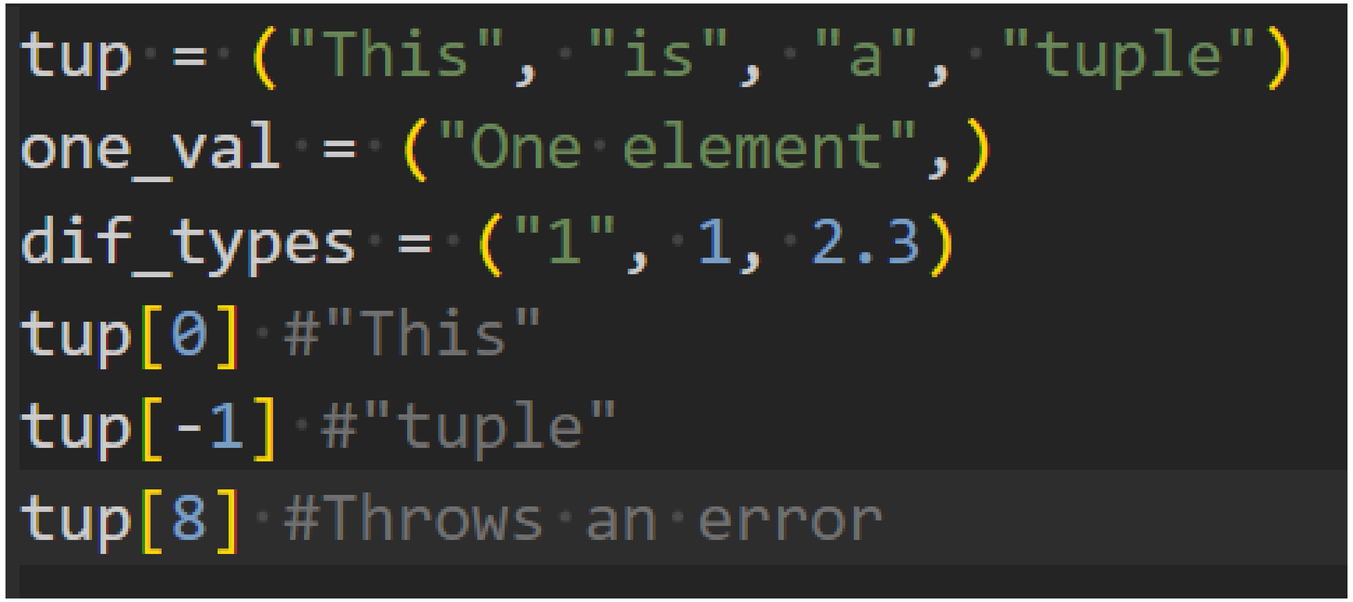
- Allows us to store data sequentially
- Each element of the list has an associated position, (0-based)
- To access the first element, of a list use `list_name[0]`, second `list_name[1]`, and so on
- Lists are defined with [ ], and can have elements listed out in them such as `[2, 3, “D”, False, 17.6]`
- Can insert, remove, and change items
- Can contain duplicate values
- Accessing out of bounds will throw an error!



```
empty_list = []
filled_list = [2, True, "Fintech", [17, 14, 2]]
filled_list[0]#2
len(filled_list)#4
filled_list[1] = False#[2, False, "Fintech", [17, 14, 2]]
filled_list.append(20)#[2, False, "Fintech", [17, 14, 2], 20]
filled_list.pop()# [False, "Fintech", [17, 14, 2], 20]
filled_list.remove(1)# [False, [17, 14, 2], 20]
filled_list.insert(1, "James")# [False, "James, [17, 14, 2], 20"]
filled_list[-1]#20
```

Tuples

- Tuples are like lists, but unchangeable (immutable)
- Values can only be accessed and not modified
- There can be duplicates
- Declared with `( )`, such as `(1, False, “k”)`



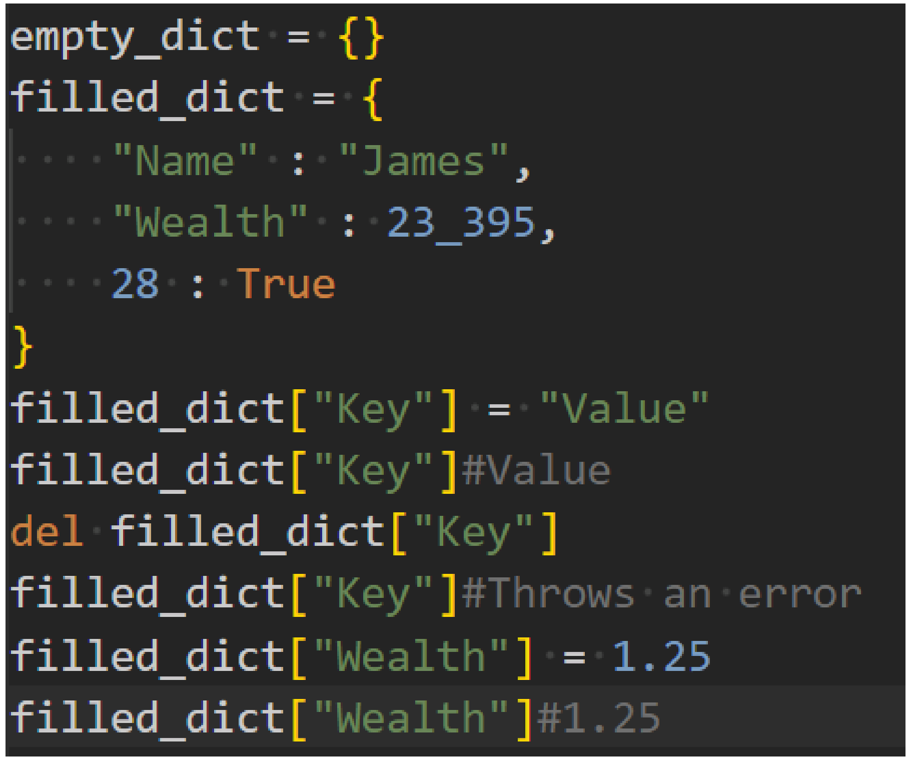
```python
tup = ("This", "is", "a", "tuple")
one_val = ("One element",)
dif_types = ("1", 1, 2.3)
tup[0] #"This"
tup[-1] #"tuple"
tup[8] #Throws an error
```

Dicts

- Declared with `{}`
- Stores elements in Key : Value pairs, separated by commas
- Access a value with its key,

`dict[“Key”] == “Value”`

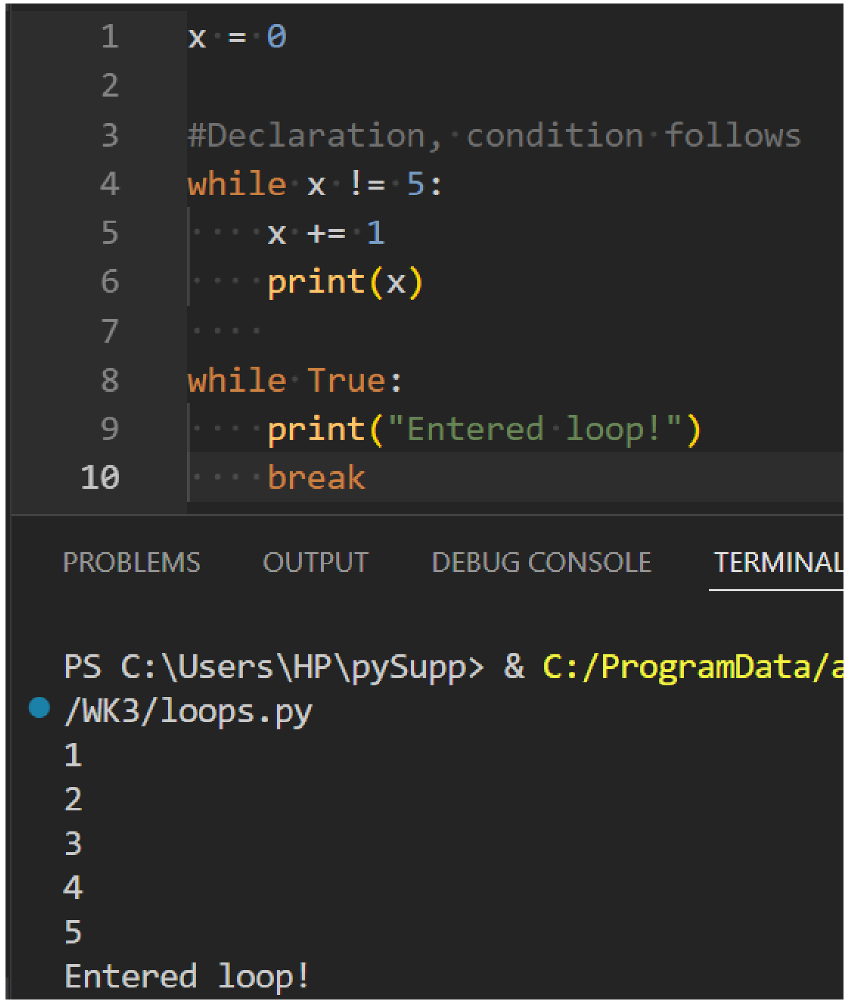
- There cannot be duplicate Keys
- Values can be updated, and Keys removed



```
empty_dict = {}
filled_dict = {
    "Name" : "James",
    "Wealth" : 23_395,
    28 : True
}
filled_dict["Key"] = "Value"
filled_dict["Key"]#Value
del filled_dict["Key"]
filled_dict["Key"]#Throws an error
filled_dict["Wealth"] = 1.25
filled_dict["Wealth"]#1.25
```

Loops - while

- What if we want to run some code more than once? Instead of typing the same command out multiple times, a loop allows us to run it until a condition is meant
- `while` loops run as long as a condition in () evaluates to `True`
- The `break` statement prematurely breaks a loop
- The `continue` statement prematurely goes to the next iteration of a loop (more useful with `for`)



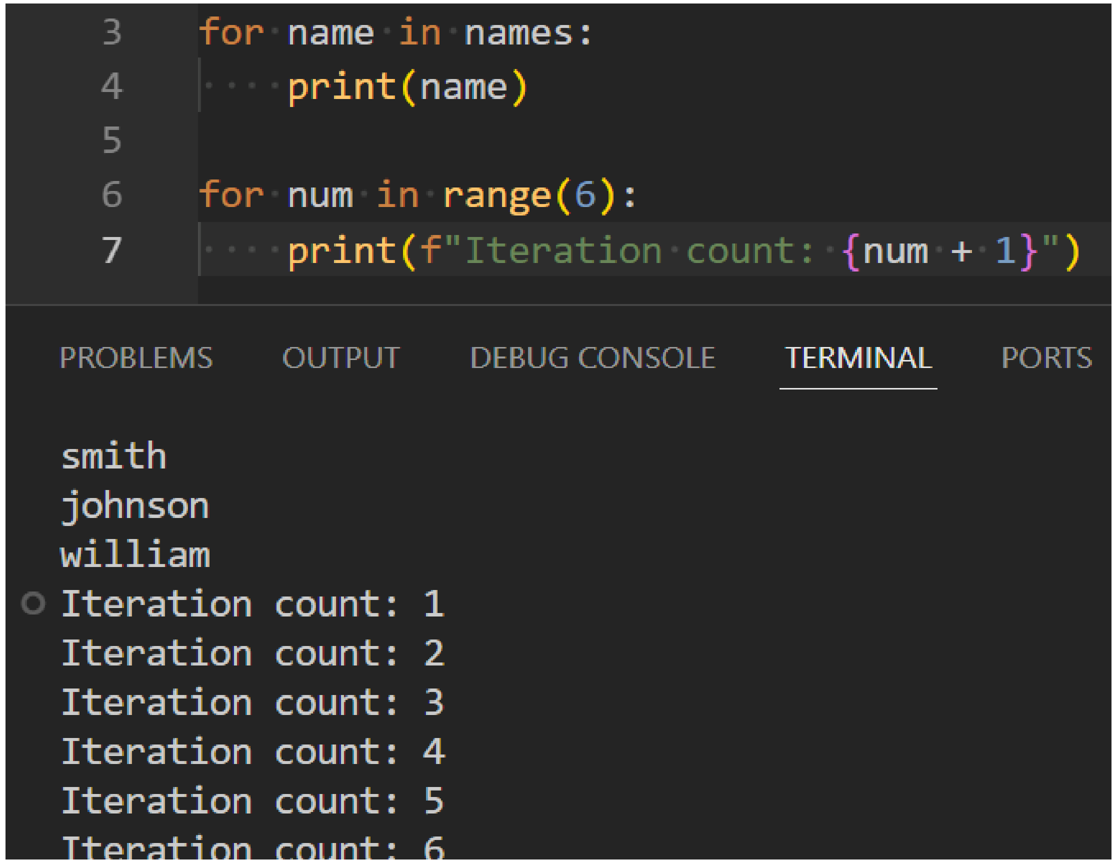
```python
x = 0

#Declaration, condition follows
while x != 5:
    x += 1
    print(x)
    
while True:
    print("Entered loop!")
    break
```

```
PS C:\Users\HP\pySupp> & C:/ProgramData/a
/WK3/loops.py
1
2
3
4
5
Entered loop!
```

Loops - for

- Increments for a bounded period of iterations
- Easily allows data structures to be looped through
- With range() function, allows for code to be run a specified amount of times
- Declares a variable to access during each iteration



```python
for name in names:
    print(name)

for num in range(6):
    print(f"Iteration count: {num + 1}")
```

```
smith
johnson
william
Iteration count: 1
Iteration count: 2
Iteration count: 3
Iteration count: 4
Iteration count: 5
Iteration count: 6
```

The End!

- Next session we will be starting to use external libraries!

- Thank you!