Composites, Tuples: Part 4

A composite type similar to a list but the elements store data that should not change.

James Tam

Terminology: Mutable, Constant, Immutable,

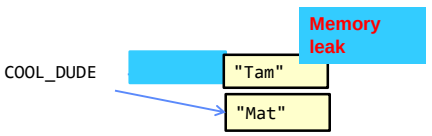
- **(New) Mutable types:**
  - The original memory location *can* change.
  - You can visualize simple types as being mutable.

```
num = 12
num = 17
```

num | 17

- **Constants:**
  - Memory location *shouldn't* change (Python): may produce a logic error if modified e.g. `GST_RATE = 0.05`
  - Memory location syntactically *cannot* change (C++, Java): produces a syntax error (violates the syntax or rule that constants cannot change)
- **(New) Immutable types:**
  - The *original* memory location *won't change*
  - Changes to a variable of a pre-existing immutable type creates a new location in memory. There are now two locations.

```
COOL_DUDE = "Tam"
COOL_DUDE = "Mat"
```



James Tam

Lists Are Mutable

- **Example**

```
aList = [1,2,3]
aList[0] = 10
print(aList)  # [10,2,3]
```

The original list can change (modifying an element) making this type mutable

James Tam

Strings Are Immutable

- Even though it may look a string can change they actually cannot be edited (original memory location cannot change).
- **Name of the example program**: `1immutableStrings.py`
  - Learning: strings are immutable:
    - Using the assignment operator in conjunction with the name of the whole string produces a new string (string variable refers to a new string not the original string).
    - Attempting to modify a string produces an error.

```
s1 = "hi"
print(s1)
s1 = "bye"      # New string created
print(s1)
s1[0] = "G"     # Error
```

```
hi
bye
```

Cannot modify the characters in a string (immutable)

```
Traceback (most recent call last):
  File "12immutableStrings.py", line 7, in <module>
    s1[0] = "G"
TypeError: 'str' object does not support item assignment
```

James Tam

New Term: Memory Leak

- (Paraphrased from different sources): A memory leak occurs when memory that has been allocated (e.g. during the creation of a: local identifier, list, object etc.) can no longer be accessed (and deallocated).
- Result: the consumption in memory may (depending upon the language) result in a run-time error or a general slow down of the device running the program.
- Example of a possible memory leak:

```
aStr1 = "hi"
aStr1 = "bye"  #String hi can become a memory leak
```

James Tam

Tuples

- Much like a list:
  - A tuple is a composite type whose elements can consist of any other type.
    - Heterogeneous: Elements do not have to be of the same type.
  - May contains values of different types.
  - Elements of a Tuple have an order and accessed via the index.
- Tuples support many of the same operators as lists such as indexing but not methods that modify it e.g. append
- Like lists each element of a tuple is not confined to characters (string of length 1).
- But unlike a list a tuple is immutable.
  - It stores data that **should not change**.
  - Elements cannot change, length cannot change.
  - 'Changes' creates a new tuple.
  - In that way it's somewhat analogous to a named constant (e.g. `PI = 3.14`) but unlike this named constant changes can only produce a new tuple.

James Tam

Creating Tuples

- **Format**:

  tuple_name = (*value*[1], *value*[2]...*value*[n])

- **Example**:

```
#Empty tuple created, address in 'tup'
tup = ()

#New tuple created, address of new tuple in 'tup'
tup = (1,2,"foo",0.3)
```

James Tam

A Small Example Using Tuples

- **Name of the online example**: `2simpleTupleExample.py`
  - Learning: accessing an entire tuple, accessing individual elements, tuples are an immutable type.

```
tup = (1,2,"foo",0.3)
print(tup)
print(tup[2])
tup[2] = "bar"
```

```
(1, 2, 'foo', 0.3)
foo
```

Error (trying to change an immutable):
"TypeError: object does not support item assignment"

James Tam

Function Return Values

- Although it appears that functions in Python can return multiple values they are in fact consistent with how functions are defined in other programming languages.
- Functions can either return zero or *exactly one value* only.
- Specifying the return value with brackets merely returns one tuple back to the caller (to be more specific it is the address of a tuple)

```
def fun():
    return(1,2,3)
```

Returns: A tuple with three elements

```
def fun(num):
    if (num > 0):
        print("pos ")
        return()
    elif (num < 0):
        print("neg")
        return()
```

Nothing is returned back to the caller (empty tuple)

James Tam

Functions Changing Multiple Items

- Because functions only return 0 or 1 items (Python returns one composite) the mechanism of **passing by reference** (covered earlier in this section) is an **important** concept.
  - What if more than one change must be communicated back to the caller (only one entity can be returned).
  - Multiple changes to parameters (>1) **must** be passed by reference.

James Tam

Proving That Python Functions Return A Tuple

- **Name of the online example**:
  3functionReturnValues.py
  - Learning:
    - Demonstrating functions return tuples
    - Iterating a tuple using loops: for, while.

```
def fun():
    tupleInFun = (1.5,2,7,0.3)
    return(tupleInFun)

def start():
    tupleInStart = fun()
    print("Iterating using a for-loop in conjunction with
      the 'in' operator")
    for element in tupleInStart:
        print("%.1f" %(element))
```

James Tam

Proving That Python Functions Return A Tuple (2)

```
    print()
    i = 0
    numElements = len(tupleInStart)
    print("Iterating using a while-loop in conjunction with" \
      +" the len() function")
    while(i < numElements):
        print("%.1f" %(tupleInStart[i]))
        i = i + 1
```

James Tam

Packing A Tuple

- **Name of the full online example:** `4packing_unpacking_tuples.py`
  - **New terminology, packing a tuple:** python encounters multiple values separated by commas it will create a new tuple to store the separate values into one composite.
    - Example:

```
tuple = 1,2,3
print(type(tuple))
```

  - **New terminology, unpacking a tuple:** a tuple has been created and a program instruction stores each element of the tuple into individual variables.
    - Example:

```
def fun():
   aTuple = (1,2.0,False)
   return(aTuple)
num1,num2,num3 = fun()
print(type(num1),type(num2),type(num3))
```

    - Note: if you do this make sure that the number of individual variables exactly matches the number of elements in the tuple or the unpacking will fail.

James Tam

Singletons And Tuples

- Sometimes a program must only have one instance of an entity e.g. the "print daemon" that manages print jobs on a computer or server.
- The Singleton 'pattern' can be applied to ensure there is no more than one instance.
- One 'approach' for ensuring Singleton pattern in python:
  - **General format:**
    - `singletonVariable = {<element>,}`
  - **Example:**
    - `#Element stores all details of print jobs on the computer`
    - `daemon = {printingDetails,}`

James Tam

Tuple Operations (Table: Zhao/Hudson)

- **Name of the full online example**: `5list_operations`

Operation	Description	Example
Index	Access single element	`aTuple[0]`
Slicing	Make a copy of part of a tuple	`aTuple[start:end]`
Concatentation	Combines elements of 2 tuples into a new, larger tuple	`bigger = tuple1 + tuple2`
Length	Determining the number of elements	`len(aTuple)`
Repetition	Repeat 'n' times the elements of one tuple (with the order retained) into a new tuple	`aTuple*3`
Membership	Determine if item is an element in a tuple	`if item in aTuple:`
Iteration	Iterate (step through) each element in a tuple	`for element in aTuple` `  print(element)`
Location (index of)	Returns index of the 1st occurrence of an element, run time error if not in list	`aTuple.index(True)`

James Tam

After This Section You Should Now Know

- What is a tuple, common operations on tuples such as creation, accessing elements, displaying a tuple or elements.
- The actual value returned from a function.
- New terminology: packing/unpacking tuples.
- Common operations for tuples.

James Tam

Copyright Notification

- Unless otherwise indicated, all images in this presentation were provided courtesy of James Tam.