

Composite Types: Other Composites

You will learn how to create new variables that are collections of other entities: strings (character composite), tuples (similar to a list but immutable)

ASCII Values (Reminder)

- Each character is assigned an ASCII code e.g., 'A' = 65, 'b' = 98
- The `chr()` function can be used to determine the character (string of length one) for a particular ASCII code (number to character)
- The `ord()` function can be used to determine the ASCII code for a 'character' - string of length one (character to number)
- **Name of the example program:** `1ascii.py`
 - Learning: converting to/from ASCII codes to the equivalent character.

```
aChar = input("Enter a character whose ASCII value that you wish to
see: ")
print("ASCII value of %s is %d" %(aChar,ord(aChar)))
aCode = int(input("Enter an ASCII code to convert to a character: "))
print("The character for ASCII code %d is %s" %(aCode,chr(aCode)))
```

```
Enter a character whose ASCII value that you wish to see: A
ASCII value of A is 65
Enter an ASCII code to convert to a character: 66
The character for ASCII code 66 is B
```

String: Composite

- Strings are just a series of characters (e.g., alpha, numeric, punctuation etc.)
 - Like a list a string is:
 - A composite type (can be treated as one entity or individual parts can be accessed).
 - **Name of example:** "2stringComposite.py"
 - Learning: strings are composite, how to access the entire composite string and how to access individual elements

```
aString1 = "hello"
print("Whole string %s" %(aString1))
print("Sub string %s-%s" %(aString1[1],aString1[4]))
```

```
Whole string hello
Sub string e-o
```

```
0 1 2 3 4
hello
```

Passing Strings As Parameters

- A string is composite so either the entire string or just a sub-string can be passed as a parameter.
- **Name of example:** 3stringParameters.py
 - Learning: How to pass a string (or substring) to a function.

```
def fun1(str1):
    print("Inside fun1 %s" %(str1))
```

```
def fun2(str2):
    print("Inside fun2 %s" %(str2))
```

```
def start():
    str1 = "abc"
    print("Inside start %s" %(str1))
    fun1(str1)
    fun2(str1[1])
```

```
Inside start abc
Inside fun1 abc
Inside fun2 b
```

Passing whole string

Passing part of a string

James Tam

Terminology: Mutable, Constant, Immutable,

- **(New) Mutable types:**

- The original memory location *can* change.

- You can visualize simple types as being mutable. num 17

num = 12

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:**

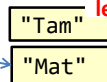
- The *original* memory location *won't* change

- 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"

COOL_DUDE



Memory
leak

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.

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:** 4immutableStrings.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
```

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
```

Substring Operations

- Sometimes you may wish to extract out a portion of a string.
 - E.g., Extract first name “James” from a full name “James T. Kirk, Captain”
- This operation is referred to as a ‘substring’ operation in many programming languages.
- There are two implementations of the substring operation in Python:
 - String slicing
 - String splitting

1 The name James T. Kirk is © CBS

String Slicing

Included
in the
slice

Excluded in the
slice

- Slicing a string will return a portion of a string based on the indices provided
- The index can indicate the start (include) and end point (exclude) of the substring.

– **Format:**

`string_name [start_index : end_index]`

- **Name of example:** 5stringSlicing.py

- Learning: how the slicing operator works

```
aString = "abcdefghij"
```

```
print(aString)
```

```
temp = aString[2:5]
```

```
print(temp)
```

```
temp = aString[:5]
```

```
print(temp)
```

```
temp = aString[7:]
```

```
print(temp)
```

```
abcdefghij
```

```
cde
```

```
abcde
```

```
hij
```

```
0 1 2 3 4 5 6 7 8 9
a b c d e f g h i j
```

From start to the
end (excluded)

From 7 (included)
until the end

Example Use: String Slicing

- Where characters at fixed positions must be extracted.
- Example: area code portion of a telephone number
"403-210-9455"
 - The "403" area code could then be passed to a data base lookup to determine the province.

James Tam

String Splitting

- Divide a string into portions with a particular character determining where the split occurs.
- Practical usage
 - The string "The cat in the hat" could be split into individual words (split occurs when spaces are encountered).
 - "The" "cat" "in" "the" "hat"
 - Each word could then be individually passed to a spell checker.

String Splitting (2)

- **Format:**

string_name.split("<character used in the split")

- **Online example:** 6stringSplitting.py

– Learning: how the slicing operator works.

```
aString = "man who smiles"
# Default split character is a space
one, two, three = aString.split()
```

```
man
who
smiles
```

```
print(one)
print(two)
print(three)
```

```
aString = "James,Tam"
```

```
James "The Bullet" Tam
```

```
first, last = aString.split(",")
nic = first + " \"The Bullet\" " + last
print(nic)
```

James Tam

String Testing Functions¹

- These functions test a string to see if a given condition has been met and return either "True" or "False" (Boolean).

- **Format:**

string_name.function_name()

¹ These functions will return false if the string is empty (less than one character).

String Testing Functions (2)¹

| Boolean Function | Description |
|------------------|--|
| isalpha() | Only true if the string consists only of alphabetic characters. |
| isdigit() | Only returns true if the string consists only of digits. |
| isalnum() | Only returns true if the string is composed only of alphabetic characters or numeric digits (alphanumeric) |
| islower() | Only returns true if the alphabetic characters in the string are all lower case. |
| isspace() | Only returns true if string consists only of whitespace characters (" ", "\n", "\t") |
| isupper() | Only returns true if the alphabetic characters in the string are all upper case. |

¹ Each one of these functions ('method') must be preceded by a string variable and a dot e.g. aStr.isalpha() #where aStr refers to a string

Applying A String Testing Function

Name of the example: 7stringTestFunctions.py

- Learning: using the isdigit() function to check for invalid types (float instead of integer)

```
ok = False
while(ok == False):
    temp = input("Enter an integer: ")
    ok = temp.isdigit()
    if(ok == False):
        print(temp, "is not an integer")
num = int(temp)
num = num + num
print(num)
```

Heuristic (end of
"loops") applied also
(good error message)

Enter an integer: abc
abc is not an integer

Enter an integer: 11.2
11.2 is not an integer

Enter an integer: 12
24

Functions That Return Modified Copies Of Strings (IF There Is Time)¹

- These functions return a modified version of an existing string (leaves the original string intact). Common whitespace characters = sp, tab, enter

| Function | Description |
|---------------------------|---|
| <code>lower()</code> | Returns a copy of the string with all the alpha characters as lower case (non-alpha characters are unaffected). |
| <code>upper()</code> | Returns a copy of the string with all the alpha characters as upper case (non-alpha characters are unaffected). |
| <code>strip()</code> | Returns a copy of the string with all leading and trailing whitespace characters removed. |
| <code>lstrip()</code> | Returns a copy of the string with all leading (left) whitespace characters removed. |
| <code>rstrip()</code> | Returns a copy of the string with all trailing (right) whitespace characters removed. |
| <code>lstrip(char)</code> | Returns a copy of the string with all leading instances of the character parameter removed. |
| <code>rstrip(char)</code> | Returns a copy of the string with all trailing instances of the character parameter removed. |

¹ Each one of this functions ('method') must be preceded by a string variable and a dot e.g. `aStr.lower()` #aStr refers to a string

Examples: Functions That Return Modified Copies (IF There Is Time)

Name of the example program: 8stringModificationFunctions.py

Learning: learning how common string functions operate

```

aString = "talk1! AbouT"
print(aString)           talk1! AbouT
aString = aString.upper()
print(aString)           TALK1! ABOUT

aString = "xxhelxlo therex"
print(aString)           xxhelxlo therex
aString = aString.lstrip("x")
print(aString)           helxlo therex
aString = "xxhellx thxr"
aString = aString.rstrip("x")
print(aString)           xxhellx thxr

```

Tuples

- Much like a list, a tuple is a composite type whose elements can consist of any other type.
- Tuples support many of the same operators as lists such as indexing.
- However tuples are immutable.
- 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**.
 - 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.

Creating Tuples

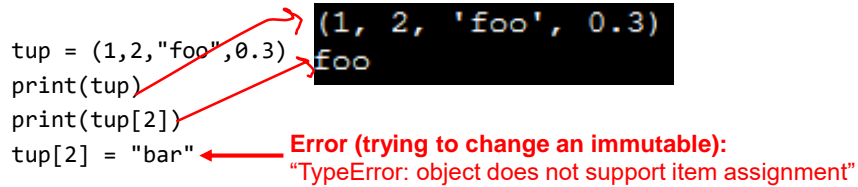
- **Format:**
`tuple_name = (value1, value2...valuen)`
- **Example:**
`tup = (1,2,"foo",0.3)`

A Small Example Using Tuples

- **Name of the online example:** 9simpleTupleExample.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"
```



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

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.

```
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)

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.

Proving That Python Functions Return A Tuple

- **Name of the online example:**
`10functionReturnValues.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

Extra Practice

String:

- Write the code that implements string operations (e.g., splitting) or string functions (e.g., determining if a string consists only of numbers)

After This Section You Should Now Know

- What is the difference between a mutable and an immutable type
- How strings are actually a composite type
- Common string functions and operations
- How a tuple is a composite, immutable type.
- Iterating tuples using for and while loops

After This Section You Should Now Know (2)

- What is a tuple, common operations on tuples such as creation, accessing elements, displaying a tuple or elements
- How functions return zero or one item