Topic/Title: **Section 3**

Section 3: Strings

***Strings are immutable and cannot be changed**

Keywords/Questions:

Notes: **3.1 String Basics** **The len() built-in function** can be used to find the length of a string (and any other sequence type).

Programs commonly access an individual character of a string. As a sequence type, every character in a string has an index, or position, starting at 0 from the leftmost character. For example, the 'A' in string 'ABC' is at index 0, 'B' is at index 1, and 'C' is at index 2.

negative indices can be used to access characters starting from the right-most character of the string,

Assign my_var with the last character in my_str. Use a negative index. my_var = my_str[-1]

Changing string variables and concatenating strings

```
string = "roses are red"
print("Original string:", string)
print("After using capitalzie:", string.capitalize())
```

Output:
Original string: roses are red
After using capitalzie: Roses are red

Convert a string to uppercase

```
# example string
string = "this should be uppercase!"
print(string.upper())
```

*A program can add new characters to the end of a string in a process known as **string concatenation.**

```
string_1 = 'abc'
string_2 = '123'

concatenated_string = string_1 + string_2
print('Easy as ' + concatenated_string)
```

***In-place modification of string variables is not allowed**
This is not allowed=>.

```
address = '900 University Ave'
address[0] = '6'
address[1] = '2'
```

Instead you'd have to write this =>

```
address = '900 University Ave'
address = '620 University Ave'
```

3.2 String formatting

A formatted string literal, or f-string, allows a programmer to create a string with placeholder expressions that are evaluated as the program executes.

```
num_items = 3
cost_taco = 1.25
print(f'{num_items} tacos cost {cost_taco * num_items}')  => 3 tacos cost 3.75
```

Additional f-string features

An = sign is provided after the expression in a replacement field to print both the expression and its result,

f'{2*4=}' produces the string "2*4=8".

{{ and }} are used to place a curly brace into an f-string. Ex: f'{{Jeff Bezos}}: => {Jeff Bezos}: Amazon

```
two_power_two = 2**2
print(f'{two_power_two=}'
```

=> two_power_two=4

```
print(f'{2*2}')  => 4
```

```
print(f'{{{2**2=}}}')  => {2**2=4}
print(f'{{2**2}}') => {2**2}
```

```
output = f'{{2}} + {{3}} = {{5}}'  =>{2} + {3} = {5}
```

Table 3.2.2: Common format specification presentation types.

Summary:

Exponent notation

```
number = 44  =>
print(f'{number:e}')
```

4.400000e+01

Fixed-point notation (programmer-defined precision)

```
number = 4
print(f'{number:.2f}')
```

4.00

Fixed-point notation (six places of precision)

```
number = 4  =>
print(f'{number:f}')
```

4.000000

Binary (integer values only)

```
number = 4
print(f'{number:b}')
```

100

Topic/Title:

Keywords/Questions:

Notes:

Adding and removing list elements

```
my_list =[10,20]          => [10,20,ABC]
my_list.append('abc')
```

```
my_list = [10, 'bw']                    => [10, 'bw']
print(my_list)
my_list.append('abc')                   => [10,'bw','abc']
print(f'After append: {my_list}')
my_list.pop(1)                          =.  [10, 'abc']
print(f'After pop: {my_list}')
```

****remove(val) removes the first element whose value matches val.**

len(list)	Find the length of the list.
list1 + list2	Produce a new list by concatenating list2 to the end of list1.
min(list)	Find the element in the list with the smallest value. All elements must be of the same type.
max(list)	Find the element in the list with the largest value. All elements must be of the same type.
sum(list)	Find the sum of all elements of a list (numbers only).
list.index(val)	Find the index of the first element in the list whose value matches val.
list.count(val)	Count the number of occurrences of the value val in the list.

3.4 Tuple basics

-Tuples store a collection of data, like a list, but is immutable – once created, the tuple's elements cannot be changed
-Supports len(), indexing, and other sequence functions.
-A new tuple is generated by creating a list of comma-separated values, such as 5, 15, 20, surrounded with parentheses, as in (5, 15, 20)
-Important, when a programmer wants to ensure that values do not change
-Tuples are typically used when element position, and not just the relative ordering of elements, is important. i.e. Lat/Long
-1st ele = latitude; 2nd ele = longitude, and the landmark will never move from those coordinates.

Named tuples

A program commonly captures collections of data like car make, model, retail price, horsepower, #seats. A **named tuple** allows the programmer to define a new simple data type that consists of named attributes. Car.price and Car.horsepower would more clearly represent a car object than a list with index positions correlating to some attributes
****namedtuple** container must be imported to create a new named tuple

namedtuple() creates only the new simple data type and does not create new data objects.

```
from collections import namedtuple
Car = namedtuple('Car', ['make','model','price','horsepower','seats'])  # Create the named tuple
```

```
chevy_blazer = Car('Chevrolet', 'Blazer', 32000, 275, 8)  # Use the named tuple to describe a car
chevy_impala = Car('Chevrolet', 'Impala', 37495, 305, 5)  # Use the named tuple to describe a different car
```

```
print(chevy_blazer)
print(chevy_impala)
```

****A data object's attributes can be accessed using dot notation, as in chevy_blazer.price.**

Create a new address object house where house.street is "221B Baker Street", house.city is "London", and house.country is "England".
house = Address('221B Baker Street', 'London', 'England')

Summary:

Topic/Title: **Set basics**

Keywords/Questions:

Notes:

A set is an unordered collection of unique elements. A set has the following properties:

1. Elements are unordered: Elements in the set do not have a position or index.
2. Elements are unique: No elements in the set share the same value.

A set can be created using the set() function, which accepts a sequence-type iterable object (list, tuple, string, etc.) whose elements are inserted into the set.

A set literal can be written using curly braces { }

```
# Create a set using the set() function.
nums1 = set([1, 2, 3])
# Create a set using a set literal.
nums2 = { 7, 8, 9 }
```

```
# Initial list contains some duplicate values
first_names = [ 'Alba', 'Hema', 'Ron', 'Alba', 'Musa', 'Ron', 'Ron' ]

# Creating a set removes any duplicate values
names_set = set(first_names)        => {'Hema', 'Ron', 'Musa', 'Alba'}
```

set() should be given a list of elements to place in the set. The provided values should be contained within a sequence-type iterable object.
So you can't do set(10, 20, 25) it would have to be set([10,20,25])

Modifying sets

Sets are mutable – elements can be added or removed using set methods.

add()
remove()
pop()

set.add(value): Add value into the set. Ex: my_set.add('abc')

set.pop(): Remove a random element from the set. Ex: my_set.pop()

Operation	Description
len(set)	Find the length (number of elements) of the set.
set1.update(set2)	Adds the elements in set2 to set1.
set.add(value)	Adds value into the set.
set.remove(value)	Removes value from the set. Raises KeyError if value is not found.
set.pop()	Removes a random element from the set.
set.clear()	Clears all elements from the set.

Operation	Description
set.intersection(set_a, set_b, set_c...)	Returns a new set containing only the elements in common between `set` and all provided sets.
set.union(set_a, set_b, set_c...)	Returns a new set containing all of the unique elements in all sets.
set.difference(set_a, set_b, set_c...)	Returns a set containing only the elements of `set` that are not found in any of the provided sets.
set_a.symmetric_difference(set_b)	Returns a set containing only elements that appear in exactly one of `set_a` or `set_b`

Summary:

Topic/Title: **3.6 Dictionary basics**

Keywords/Questions:

Notes:

A **dictionary** is a Python container used to describe associative relationships.
A dictionary is represented by the **dict** object type
A key is a term that can be located in a dictionary, such as the word "cat"
A value describes some data associated with a key, such as a definition.

A dict object is created using curly braces { } to surround the key:value pairs

players = {'Lionel Messi': 10, 'Cristiano Ronaldo': 7}

**two keys: 'Lionel Messi' and 'Cristiano Ronaldo'
**associated with the values 10 and 7

An empty dictionary is created with the expression players = { }.

Examples: last names and addresses car models and price student ID number and university email address

Adding, modifying, and removing dictionary entries

Adding: **A dictionary does not use the append() function.**
dict[k] = v: Adds the new key-value pair k-v, if dict[k] does not already exist.
Example: students['John'] = 'A+'

Modifying
dict[k] = v: Updates the existing entry dict[k], if dict[k] already exists.
Example: students['Jessica'] = 'A+'

Removing
del dict[k]: Deletes the entry dict[k].
Example: del students['Rachel']

Type	Notes
string	Sequence type: Used for text.
list	Sequence type: A mutable container with ordered elements.
tuple	Sequence type: An immutable container with ordered elements.
set	Set type: A mutable container with unordered and unique elements.
dict	Mapping type: A container with key-values associated elements.

3.9 Type conversions

A **type conversion** is a conversion of one type to another, such as an integer to a float

An implicit conversion is a type conversion automatically made by the interpreter, usually between numeric types

1 + 2 returns an integer type.
1 + 2.0 returns a float type.
1.0 + 2.0 returns a float type.

integer-to-float conversion is straightforward: 25 becomes 25.0.
float-to-integer conversion just drops the fraction: 4.9 becomes 4.

Summary: