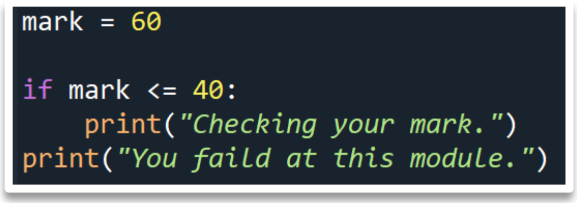
CMP 4266
Computer Programming

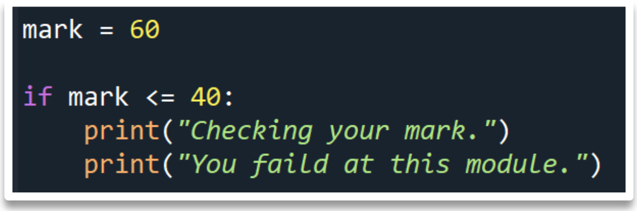
Lecture – 5

Why formatting so important?

- For syntax, your program will work differently:

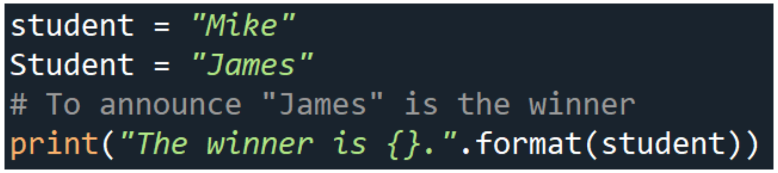


```
mark = 60

if mark <= 40:
    print("Checking your mark.")
print("You faild at this module.")
```



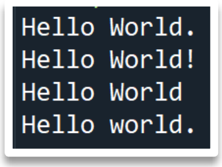
```
mark = 60

if mark <= 40:
    print("Checking your mark.")
    print("You faild at this module.")
```

- For letter case, your program will work differently:

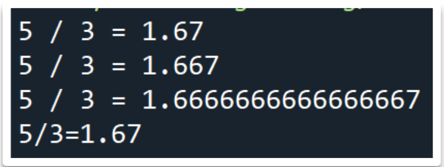


```
student = "Mike"
Student = "James"
# To announce "James" is the winner
print("The winner is {}.".format(student))
```

- For output, why should I bother?

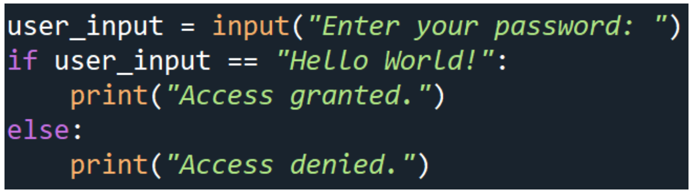


```
Hello World.
Hello World!
Hello World
Hello world.
```



```
5 / 3 = 1.67
5 / 3 = 1.667
5 / 3 = 1.6666666666666667
5/3=1.67
```

  – You might not get the expected behaviour from the program:



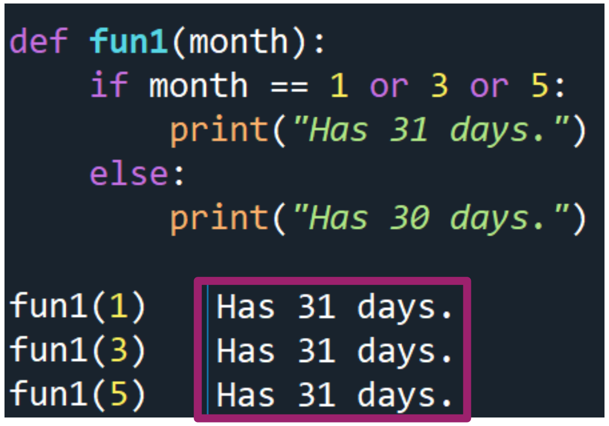
```
user_input = input("Enter your password: ")
if user_input == "Hello World!":
    print("Access granted.")
else:
    print("Access denied.")
```

  – It could lead to big lost:

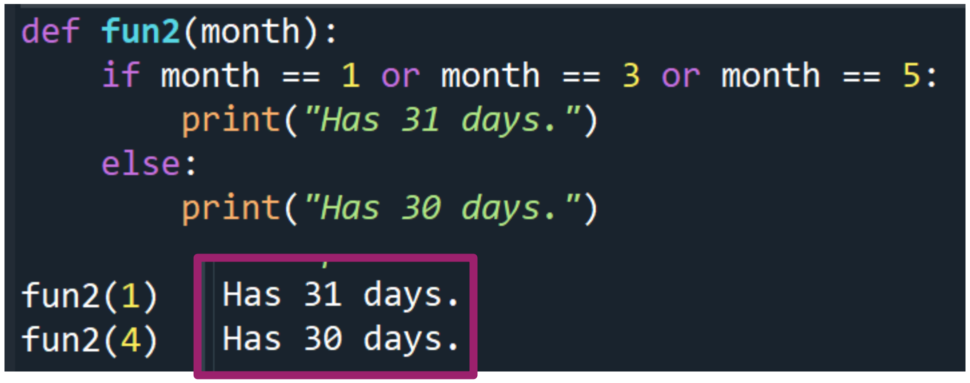
    - 10 Times in History When Misspellings Cost Big Money

A common issue: checking if condition

- Does this program work?



```
def fun1(month):
    if month == 1 or 3 or 5:
        print("Has 31 days.")
    else:
        print("Has 30 days.")

fun1(1)    Has 31 days.
fun1(3)    Has 31 days.
fun1(5)    Has 31 days.
```



```
def fun2(month):
    if month == 1 or month == 3 or month == 5:
        print("Has 31 days.")
    else:
        print("Has 30 days.")

fun2(1)    Has 31 days.
fun2(4)    Has 30 days.
```

- What will `fun1(4)` print?
  - Why it's still "`Has 31 days`."
  - Short answer:
    - A boolean expression should include a relational operator.
    - A expression such as "3" is always True.
  - Read more: ***Lecture 2, slide 44, 50-52.***

List

Introduction to Lists

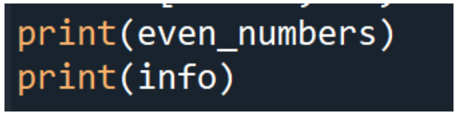
- List: an object that contains multiple data items.
  - Element: An item in a list
  - Format: `list = [item1, item2, etc.]`

- Examples
  - List of integers :

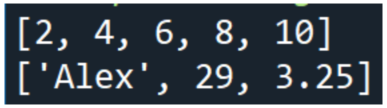
    ```
    even_numbers = [2, 4, 6, 8, 10]
    ```
  - List of Strings :

    ```
    colours = ["red", "green", "blue", "yellow"]
    ```
  - List of mixed types :

    ```
    info = ["Alex", 29, 3.25]
    ```
- `print()` function can be used to display an entire list.



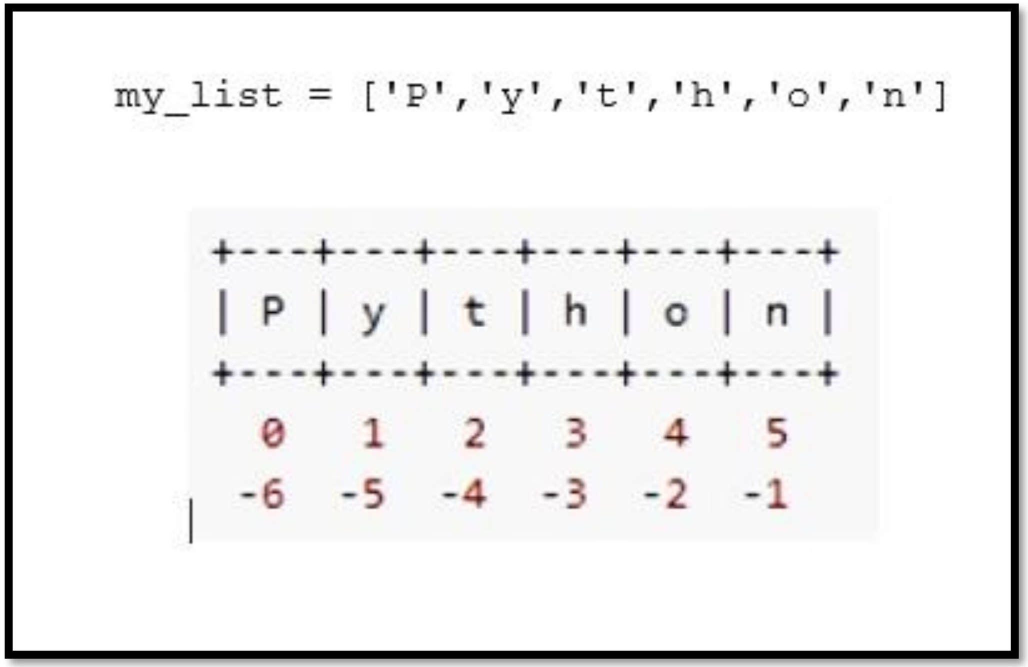
```
print(even_numbers)
print(info)
```



```
[2, 4, 6, 8, 10]
['Alex', 29, 3.25]
```

List indexing

- Index: a number specifying the position of an element in a list.
  - Enables access to individual element in list.
  - Index of the first element in the list is 0, and the second element is 1.
    - n'th element is n-1.
  - Negative indexes identify positions relative to the end of the list.
    - The index -1 identifies the last element, -2 identifies the next to last element, etc.

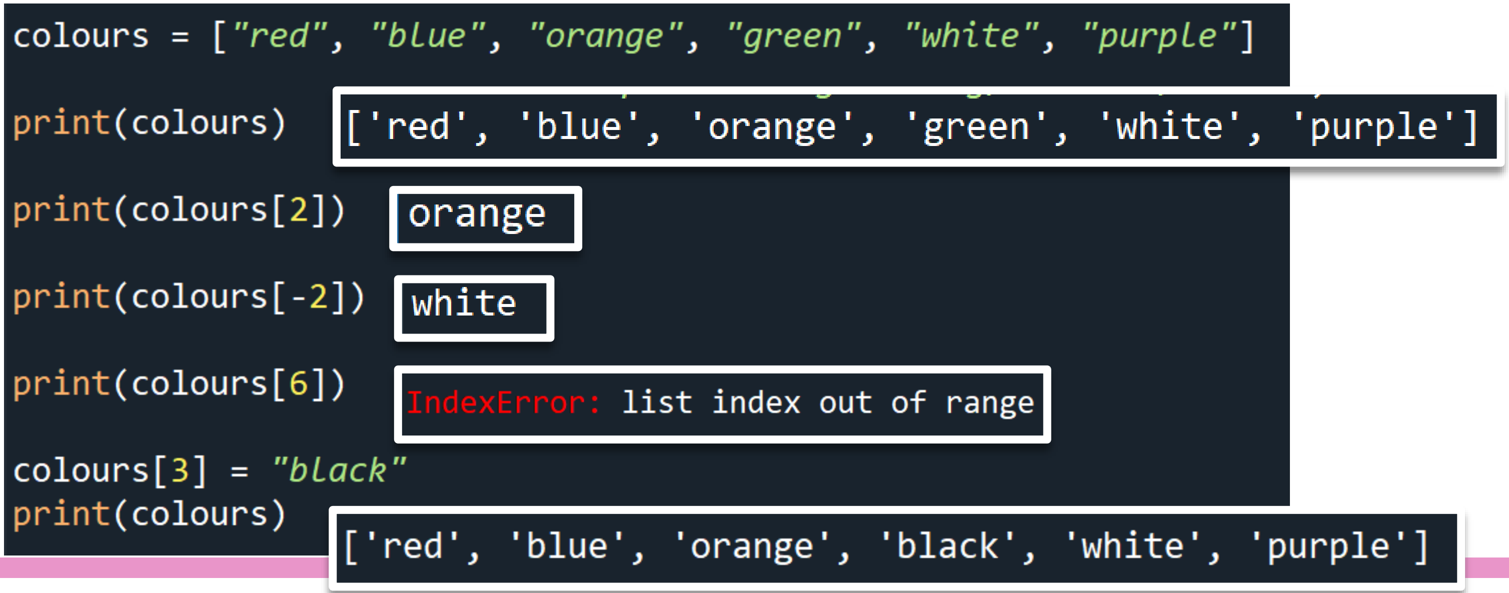


```
my_list = ['P','y','t','h','o','n']
```

```
+---+---+---+---+---+---+
| P | y | t | h | o | n |
+---+---+---+---+---+---+
  0   1   2   3   4   5
 -6  -5  -4  -3  -2  -1
```

List operation: Accessing

- To access an element in the list, use [i] where i is the index of the element.
  - `list[i]` refers to the `(i-1)` th element in the list.
  - An `IndexError` exception is raised if an invalid index is used.
- Lists are **mutable**, which means their elements can be changed.
  – `list[i] = new_value` can be used to assign a new value to a list element.
  – Must use a valid index to prevent raising of an IndexError exception.
- Example

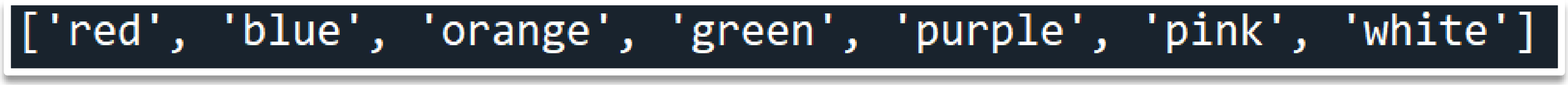


```
colours = ["red", "blue", "orange", "green", "white", "purple"]

print(colours)          ['red', 'blue', 'orange', 'green', 'white', 'purple']

print(colours[2])       orange

print(colours[-2])      white

print(colours[6])       IndexError: list index out of range

colours[3] = "black"
print(colours)          ['red', 'blue', 'orange', 'black', 'white', 'purple']
```

List operation: Concatenation

- Concatenate: join two things together.
  - The + operator can be used to concatenate two lists.
  - Example

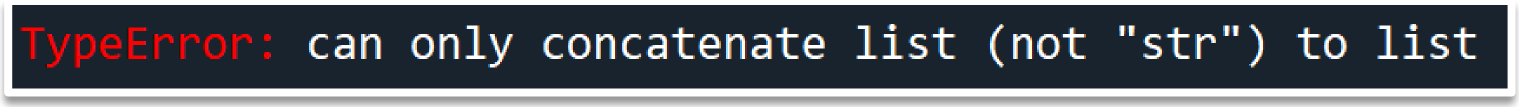
```
colours1 = ["red", "blue", "orange", "green"]
colours2 = ["purple", "pink", "white"]
colours = colours1 + colours2
print(colours)
```



```
['red', 'blue', 'orange', 'green', 'purple', 'pink', 'white']
```

- **Cannot** concatenate a list with another data type.
  - Example

```
colours = colours1 + "yellow"
```

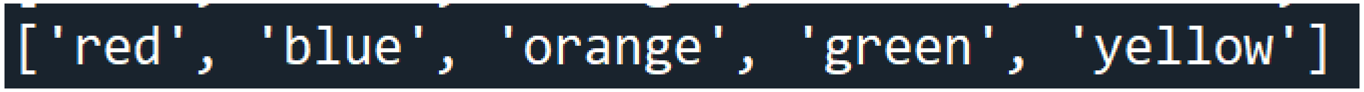


```
TypeError: can only concatenate list (not "str") to list
```

List operation: Insert an element

- How can we add "yellow" to the colours list?

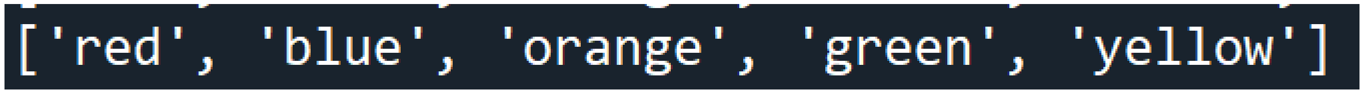
```
colours1 = ["red", "blue", "orange", "green"]


colours2 = ["yellow"]
colours = colours1 + colours2
print(colours)
```



```
['red', 'blue', 'orange', 'green', 'yellow']
```

- Use the list built-in method: `.append()`

```
colours1.append("yellow")
print(colours1)
```



```
['red', 'blue', 'orange', 'green', 'yellow']
```

List operation: Slicing

- Slice: a span of items that are taken from a sequence.
  - List slicing format: `list[start : end]`
  - Span is a sublist containing copies of elements from *start* (inclusive) to *end* (exclusive).
    - If *start* not specified, `0` is used for start index.
    - If *end* not specified, it includes all element till the end.

Example:
colours = ['red', 'blue', 'orange', 'green', 'purple', 'pink', 'white', 'black']

colours[:2]

['red', 'blue']

colours[3:7]

['green', 'purple', 'pink', 'white']

colours[5:]

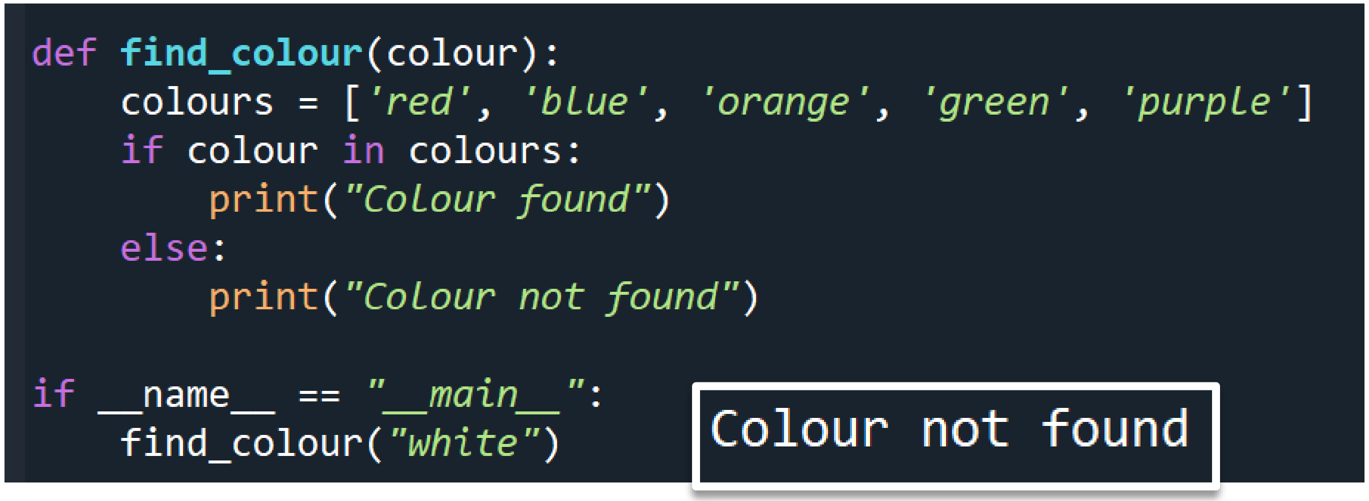
['pink', 'white', 'black']

colours[4:8]

['purple', 'pink', 'white', 'black']

List operation: in operator

- You can use the `in` operator to determine **whether an item is contained in a list.**
  - General format: *`item`* `in` *`list`*
  - Returns `True` if the item is in the list, or `False` if it is not in the list.
- Example



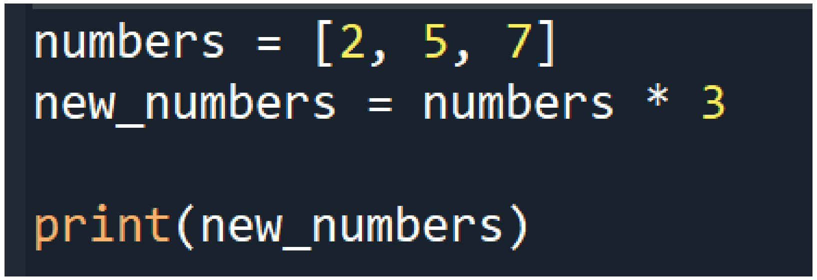
```python
def find_colour(colour):
    colours = ['red', 'blue', 'orange', 'green', 'purple']
    if colour in colours:
        print("Colour found")
    else:
        print("Colour not found")

if __name__ == "__main__":
    find_colour("white")
```

```
Colour not found
```

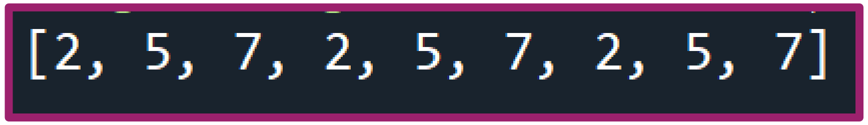
- Similarly, you can use the `not in` operator to determine whether an item is not in a list.

List operation: repetition

- Repetition operator: makes multiple copies of a list and joins them together
  - The `*` symbol is a repetition operator when applied to a sequence and an integer
    - Sequence is left operand, number is right
  - General format: *`list * n`*
- Example



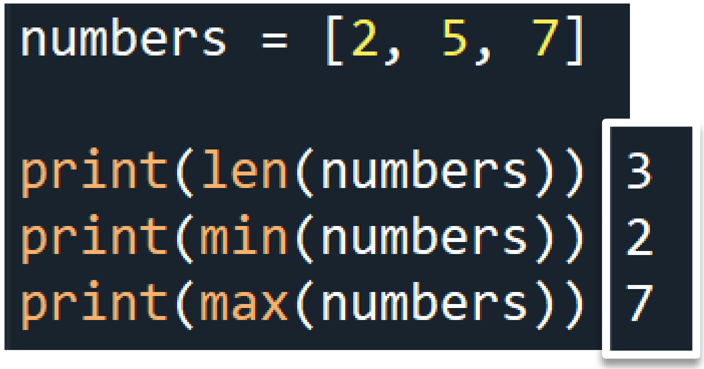
```
numbers = [2, 5, 7]
new_numbers = numbers * 3

print(new_numbers)
```



```
[2, 5, 7, 2, 5, 7, 2, 5, 7]
```

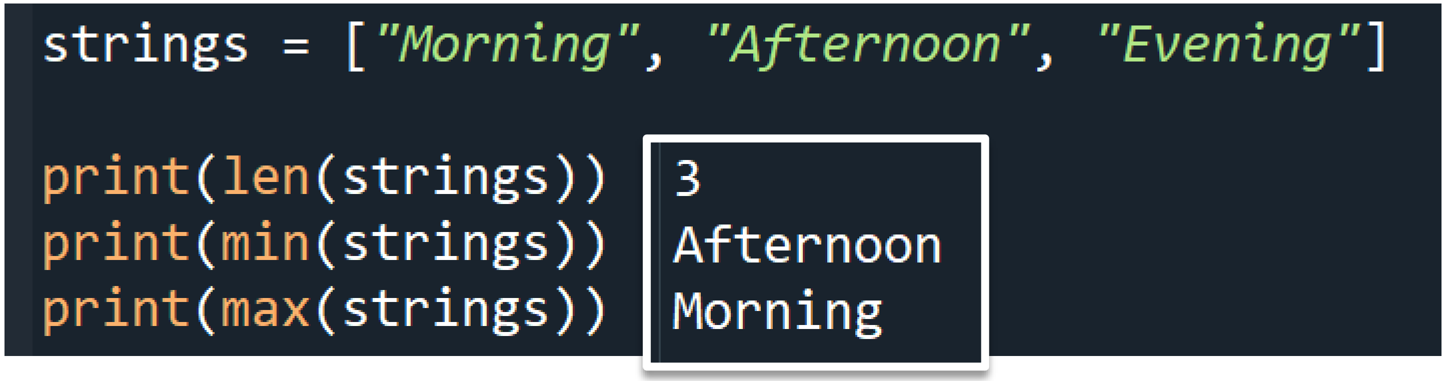
List operation: Python built-in functions

- There are three built-in functions commonly used on list:
  - `len(), min(), max()`



```python
numbers = [2, 5, 7]

print(len(numbers))
print(min(numbers))
print(max(numbers))
```

```
3
2
7
```



```python
strings = ["Morning", "Afternoon", "Evening"]

print(len(strings))
print(min(strings))
print(max(strings))
```

```
3
Afternoon
Morning
```

List operation: built-in methods

- *append(item)* – adds item to the end of the list.
  - Example

```
colours = ["red", "green", "blue"]
colours.append("white")
```

```
['red', 'green', 'blue', 'white']
```

- *index(item)* - returns the index of the first element whose value is equal to item. A ValueError exception is raised if item is not found in the list.
  - Example

```
colours = ["red", "green", "blue"]
my_index = colours.index("blue")
```

```
2
```

List operation: built-in methods 2

- *remove(item)* – removes the first occurrence of item from the list.
  - A ValueErrorexception is raised if item is not found in the list.
  - Example

```
colours = ["red", "green", "blue"]
colours.remove("green")
```

```
['red', 'blue', 'white']
```

- *reverse()* – reverses the order of the items in the list.
  - Example

```
colours = ["red", "green", "blue"]
colours.reverse()
```

```
[blue', 'green', 'red']
```

  - Note. This function does not returns anything but reverse the list in place, i.e., the item order will be changed in the original list.

List Functions

- Here is the set of functions that you can use with lists

Function	Description
L.append(x)	Add element x to the end of list L
L.extend(L2)	Add all elements of list L2 to the end of list L
L.insert(i, x)	Insert item x at the defined index i of list L
L.remove(x)	Removes item x from list L (valueError exception will be thrown if x does not exist)
L.pop(i)	Removes and returns the element at index i of list L. If no parameter is passed, the last item in L will be removed and returned

List Functions

- Here is the set of functions that you can use with lists

Function	Description
L.clear()	Removes all items from list L
L.index(x)	Returns the index of the first matched item x in list L
L.count(x)	Returns the count of times item x appears in list L
L.sort()	Sort items in list L in an ascending order
L.reverse()	Reverses the order of items in list L
L.copy()	Returns a copy of list L (*making any change to the returned list will not impact the original list L*)

Using range to iterate a list

```python
values = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
squares = []

for i in values:
    squares.append(i ** 2)

print("Initial values: {}".format(values))
print("Squares: {}".format(squares))
```

```python
values = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
squares = []

for i in range(len(values)):
    squares.append(values[i] ** 2)

print("Initial values: {}".format(values))
print("Squares: {}".format(squares))
```

5.1 Exercise – Using loops and list
10 minutes

In-class Exercise

Write a program that:

- asks the user to input() names one at a time
- adds each new name to a list called friends
- after each new name is added prints the list in alphabetical order
- The program should loop until the user types “Quit”

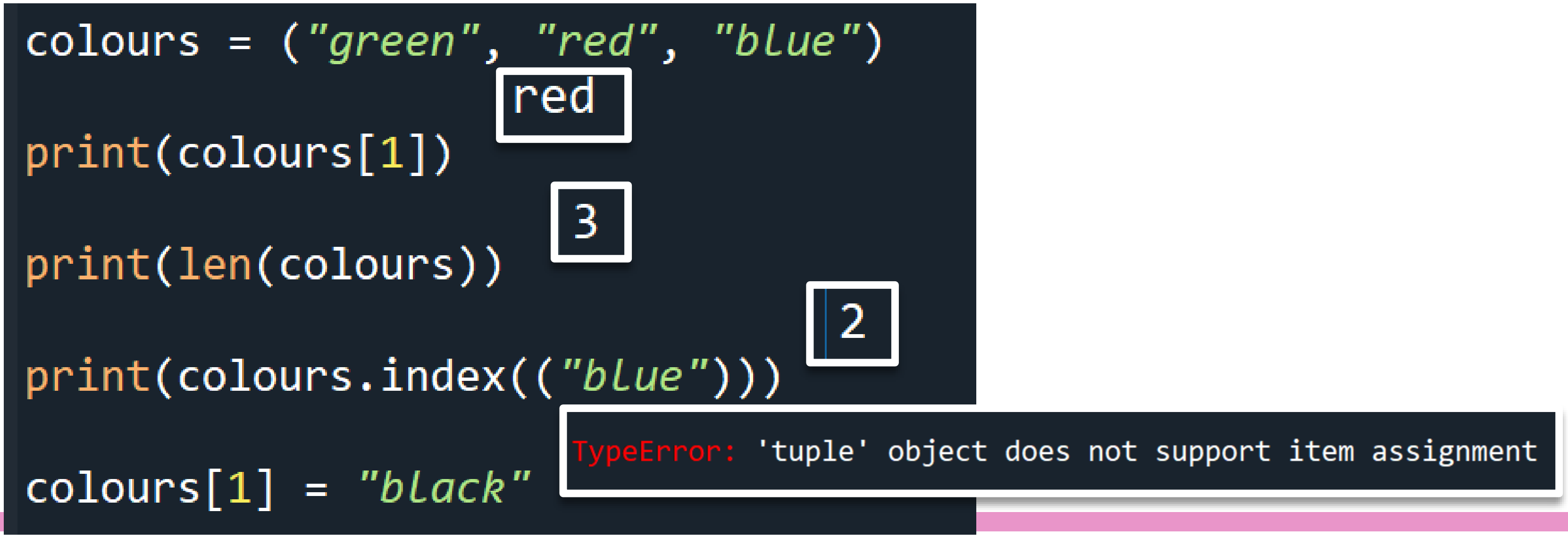
Tuple

Tuples

- <u>Tuple</u>: it's **immutable**, which means once it is created it cannot be changed.
  - Format: `tuple_name = (item1, item2,..)`
  - Tuples support similar operations as lists
    - Operations such as accessing, slicing.
    - Built in functions such as `len(), min(), max()`.
    - Methods such as `index`.

How about
`append(), remove(), reverse()`?

- Example



```
colours = ("green", "red", "blue")

print(colours[1])

print(len(colours))

print(colours.index(("blue")))

colours[1] = "black"
```

```
red
```

```
3
```

```
2
```

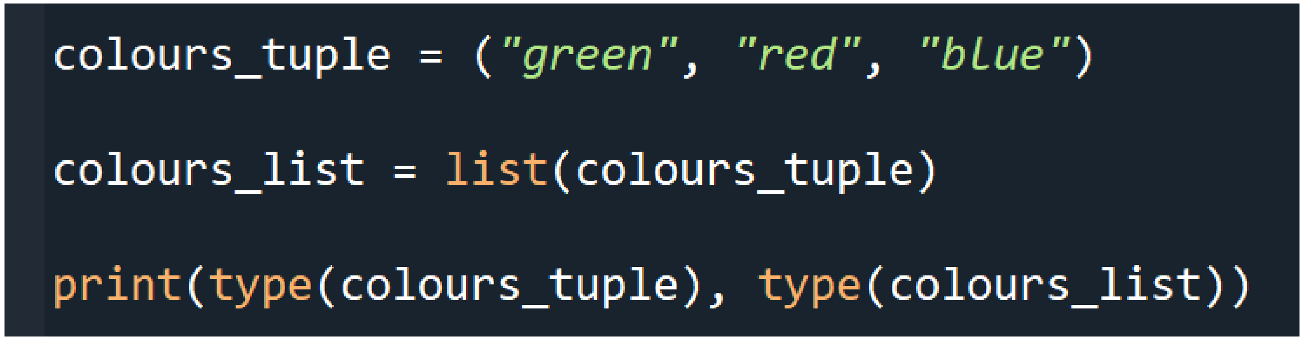
```
TypeError: 'tuple' object does not support item assignment
```

Tuples

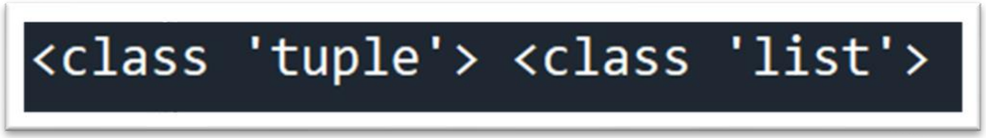
- Advantages for using tuples over lists:
  - Processing tuples is faster than processing lists.
  - Tuples are safe.

- `list()` function: converts tuple to list.
- `tuple()` function: converts list to tuple.

- Example



```python
colours_tuple = ("green", "red", "blue")

colours_list = list(colours_tuple)

print(type(colours_tuple), type(colours_list))
```



```
<class 'tuple'> <class 'list'>
```

Dictionary

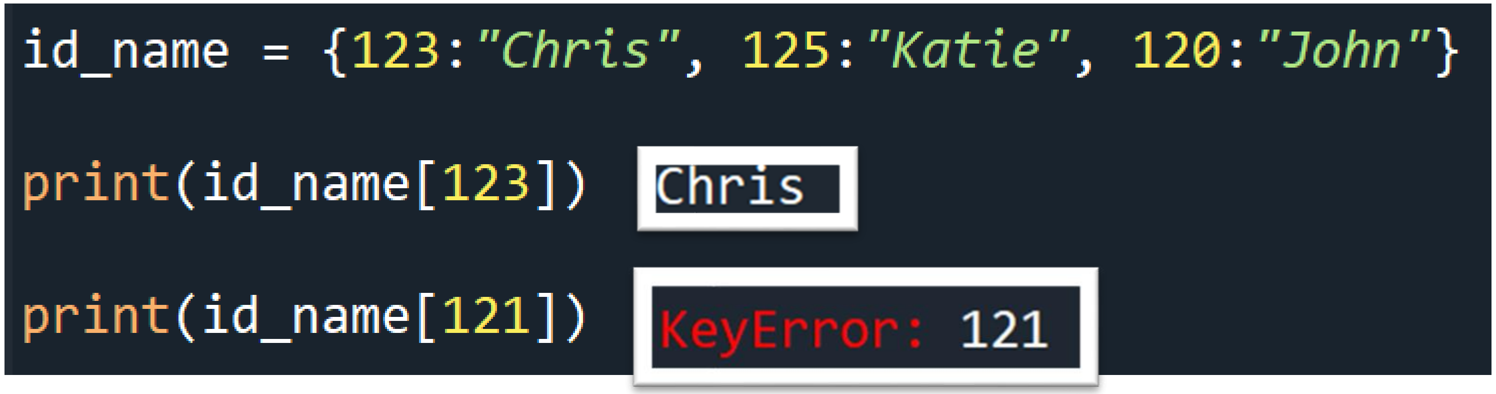
Dictionaries

- Dictionary: object that stores a collection of data
  - Each element consists of a *key* and a *value*.
    - Often referred to as *mapping* of key to value.
    - Key must be an immutable object, or for now, use either **integer** or **string**.
  - Format for creating a dictionary:
    - `dictionary = {key1:val1, key2:val2}`
  - Example:
    - `id_name = {123:"Chris", 125:"Katie", 120:"John"}`

Retrieving a value from a dictionary

- General format for retrieving value from dictionary:
  `dictionary[key]`
  - If `key` in the dictionary, associated value is returned,
  - otherwise, `KeyError` exception is raised.

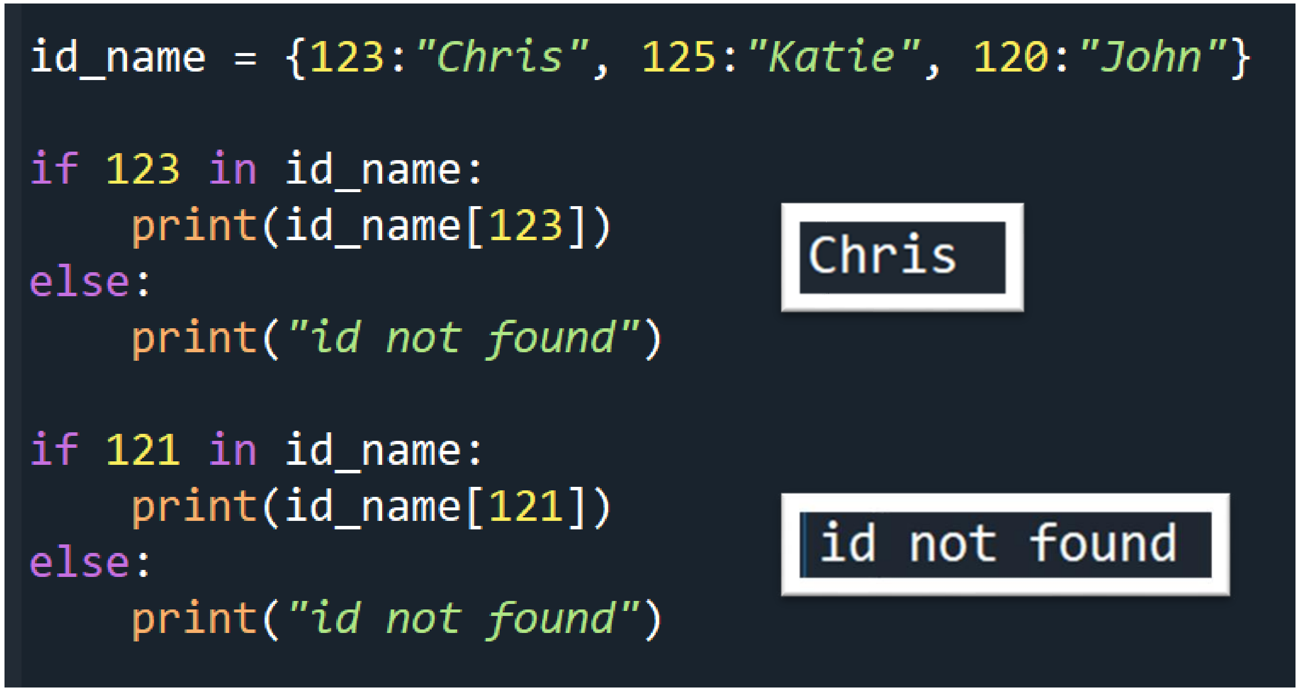
  - Example



```
id_name = {123:"Chris", 125:"Katie", 120:"John"}

print(id_name[123])    Chris

print(id_name[121])    KeyError: 121
```

Retrieving a value from a dictionary

- Test whether a key is in a dictionary using the `in` and `not in` operators.
  - Helps prevent `KeyError` exceptions.
  - Example



```
id_name = {123:"Chris", 125:"Katie", 120:"John"}

if 123 in id_name:
    print(id_name[123])
else:
    print("id not found")

if 121 in id_name:
    print(id_name[121])
else:
    print("id not found")
```

```
Chris
```

```
id not found
```

Adding elements to an existing dictionary

- Dictionaries are **mutable** objects.
  - To add a new key-value pair:

```
dictionary[key] = value
```

```
id_name = {120:"John", 123:"Chris", 125:"Katie"}

id_name[133] = "Paul"

{120: "John", 123: "Chris", 125: "Katie", 133: "Paul"}
```

  - If key exists in the dictionary, the value associated with it will be changed.

```
id_name[125] = "Robert"

{120: "John", 123: "Chris", 125: "Robert", 133: "Paul"}}
```

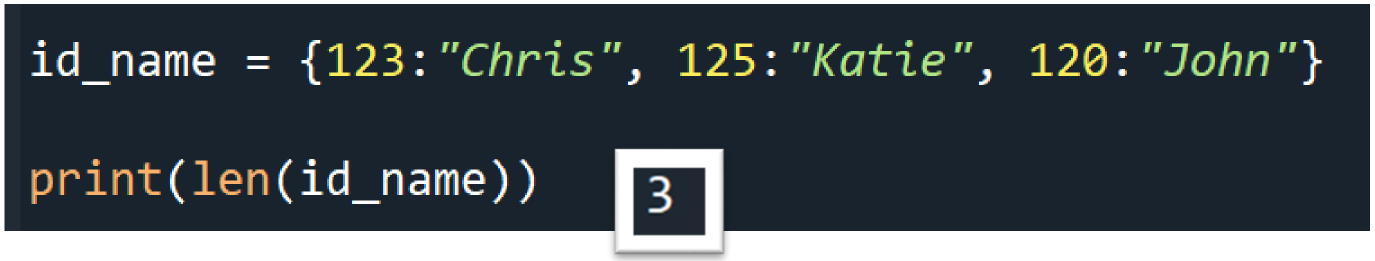
- To delete a key-value pair:

```
del dictionary[key]
```

  - If key is not in the dictionary, `KeyError` exception is raised.

More about dictionary

- `len()` function: to obtain number of elements in a dictionary.
  - Example



```
id_name = {123:"Chris", 125:"Katie", 120:"John"}

print(len(id_name))
```

```
3
```

- Values stored in a single dictionary can be of different types.
  - Example

    employee = {"name" : "Kevin Smith", "id" : 12345, "pay_rate": 25.75 }

Dictionary Functions

- Many other functions can also be used with dictionaries

Function	Description
dic.clear()	Removes all the elements from dictionary dic
dic.copy()	Returns a copy of dictionary dic
dic.items()	Returns a list containing a tuple for each key-value pair in dictionary dic
dic.get(k)	Returns the value of the specified key k from dictionary dic
dic.keys()	Returns a list containing all the keys of dictionary dic
dic.pop(k)	Removes the element with the specified key k from dictionary dic
dic.popitem()	Removes the last inserted key-value pair in dictionary dic
dic.values()	Returns a list of all the values in dictionary dic

Comparison between list and dictionary

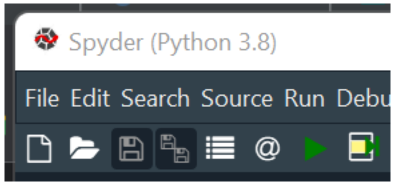
- Apart from their syntax, the key difference is the key:
  - In lists, element’s index is used as key:

0	1	2
Chris	Katie	John

  - In dictionaries keys are explicitly assigned:

123	125	120
Chris	Katie	John

- Wait, I heard that **dictionary is not sorted but list is**!
  - Dictionary is sorted from Python 3.7+.
  - Python 3.7 was released on June 27, 2018.
  - Further reading



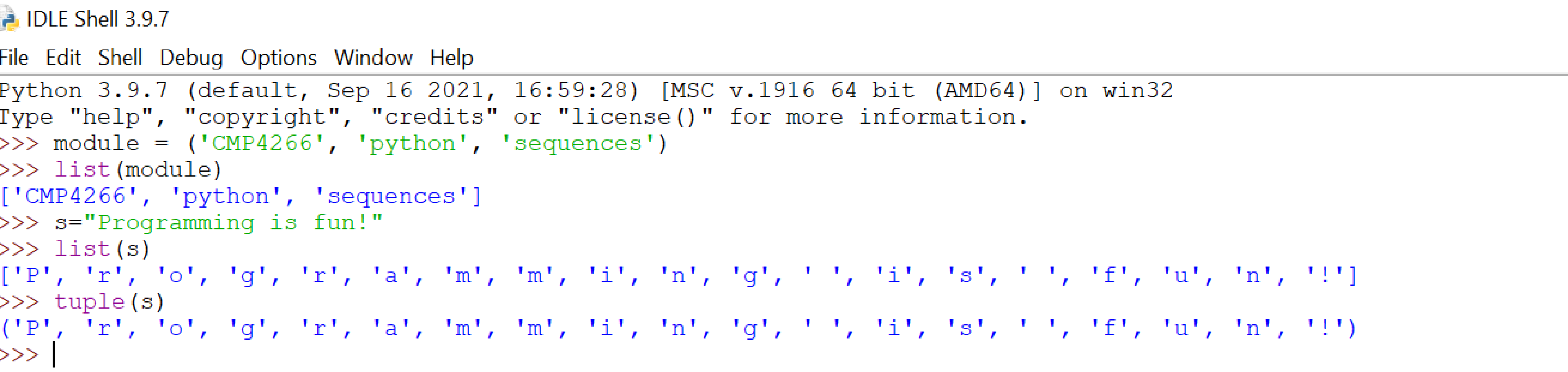
Comparison between list, tuple, dictionary

- Common:
  - They are all of sequence type in Python.
  - They are all like a container.

- Difference:
  - List and dictionary are mutable.
  - Tuple is immutable.

Converting between Sequence Types

- Like converting between integers and floating-point types, we can convert (cast) lists to tuples and vice versa.
- Use list(),and tuple()

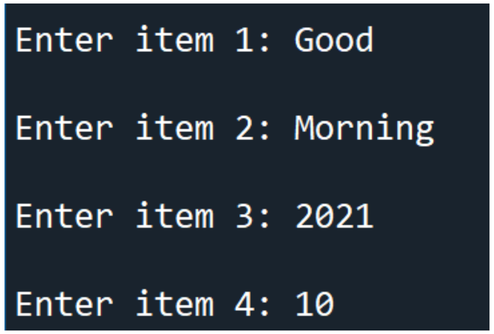


```
Python 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> module = ('CMP4266', 'python', 'sequences')
>>> list(module)
['CMP4266', 'python', 'sequences']
>>> s="Programming is fun!"
>>> list(s)
['P', 'r', 'o', 'g', 'r', 'a', 'm', 'm', 'i', 'n', 'g', ' ', 'i', 's', ' ', 'f', 'u', 'n', '!']
>>> tuple(s)
('P', 'r', 'o', 'g', 'r', 'a', 'm', 'm', 'i', 'n', 'g', ' ', 'i', 's', ' ', 'f', 'u', 'n', '!')
>>> 
```

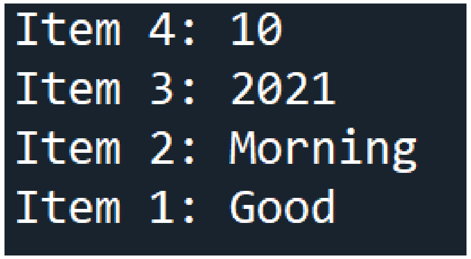
5.2 Exercise – Using loops and list
10 minutes

In-class Exercise

Write a program which takes 4 user inputs and then prints them out in the reversed order.



```
Enter item 1: Good

Enter item 2: Morning

Enter item 3: 2021

Enter item 4: 10
```



```
Item 4: 10
Item 3: 2021
Item 2: Morning
Item 1: Good
```

How many ways
can you write the codes?

5.3 Exercise – Using loops and list
10 minutes

In-class Exercise

Write a program that:

- Ask the user for the number of students
- create a dictionary that contains student id and marks obtained for all the students.

Essential Information

- Sequence type in Python
  - List, tuple, dictionary.
- List operations
  - Indexing and accessing.
  - Python built-in functions.
  - List built-in methods
- While loop
- For loop
  - range()
- Nested loops
- Make sure you have tried out all the codes in the lecture notes.

Thank you!