Introduction to Python for Image Processing

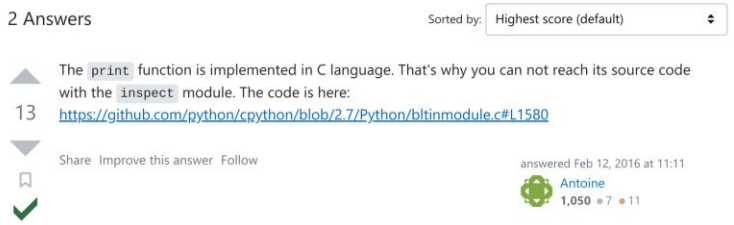
Instructor:
Junyu Chen (Day1 & Day2)
Prof. Marvin Doyley (Day3)

Functions

- Example – projector
  - a projector is a black box
  - do not know how it works
  - know the interface: input/output
  - connect any electronic to it that can communicate
  - with that input
  - black box somehow converts image from input source
  - to a wall, magnifying it
  - ABSTRACTION IDEA: do not need to know how projector works to use it

```
help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.std
out.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```



https://stackoverflow.com/questions/35360988/how-to-get-source-code-of-python-print-function

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

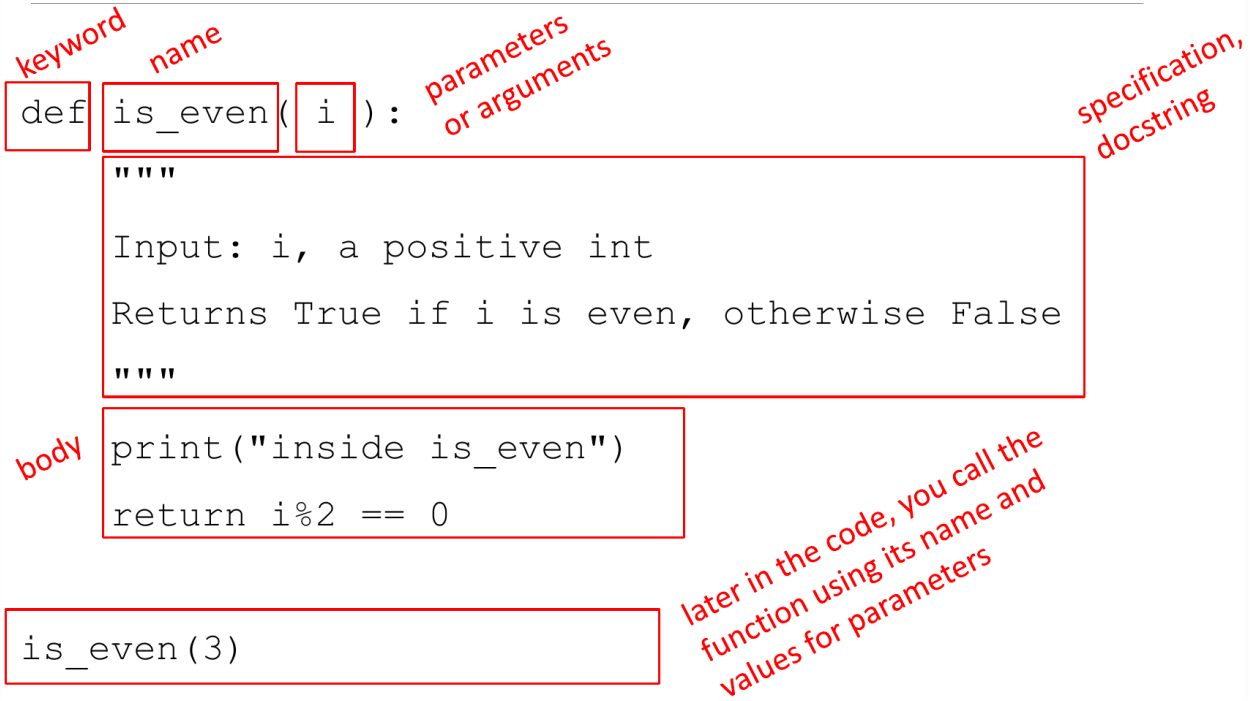
Functions

- write reusable pieces/chunks of code, called functions
- functions are not run in a program until they are "called" or "invoked" in a program
- function characteristics:
  - has a name
  - has parameters (0 or more)
  - has a docstring (optional but recommended)
  - has a body
  - returns something (0 or more)

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Functions

HOW TO WRITE and CALL/INVOKE A FUNCTION



Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Functions

IN THE FUNCTION BODY

```
def is_even( i ):
    """
    Input: i, a positive int
    Returns True if i is even, otherwise False
    """
    print("inside is_even")
    return i%2 == 0
```

run some commands

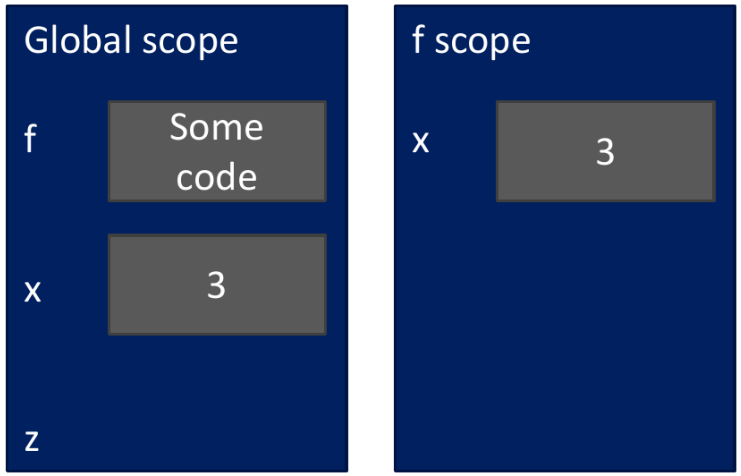
keyword

expression to evaluate and return

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Variable Scope

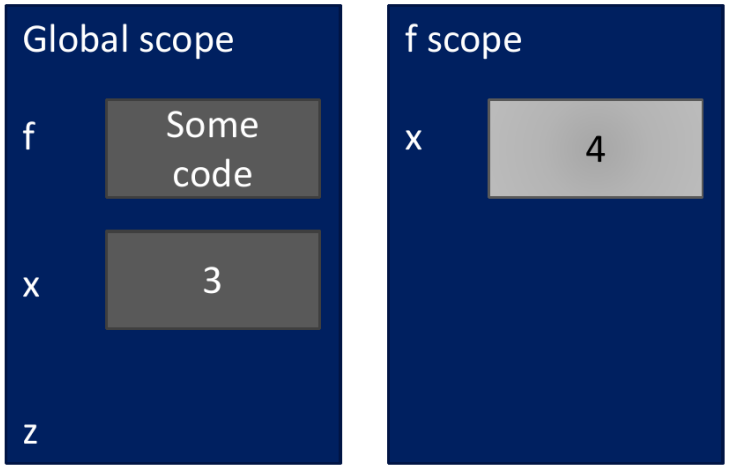
```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```



Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

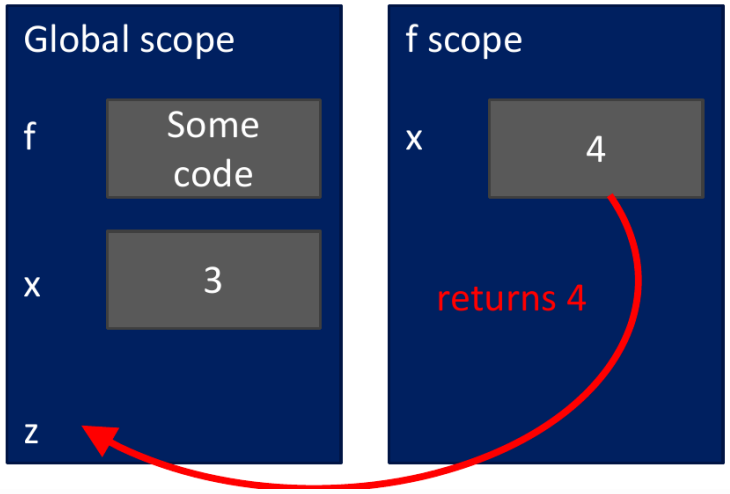
Variable Scope

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```



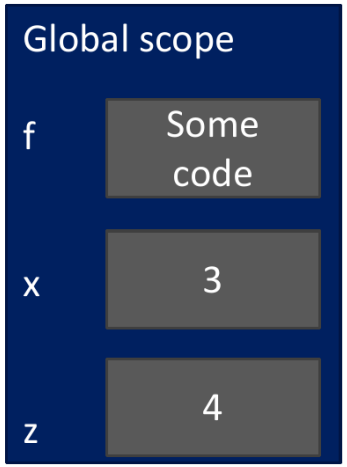
Variable Scope

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```



Variable Scope

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```



Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Scope Example

- inside a function, **can access** a variable defined outside
- inside a function, **cannot modify** a variable defined outside -- can using **global variables**, but frowned upon

```
def f(y):
    x = 1
    x += 1
    print(x)

x = 5
f(x)
print(x)
```

x is re-defined in scope of f

different x objects

```
def g(y):
    print(x)
    print(x + 1)

x = 5
g(x)
print(x)
```

x from outside g

x inside g is picked up from scope that called function g

```
def h(y):
    x += 1

x = 5
h(x)
print(x)
```

UnboundLocalError: local variable 'x' referenced before assignment

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

- http://www.pythontutor.com/

HARDER SCOPE EXAMPLE

Python Tutor is your best friend to help sort this out!

http://www.pythontutor.com/

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

If No return is Given:

```
def is_even( i ):
    """
    Input: i, a positive int
    Does not return anything
    """
    i%2 == 0
```

without a return statement

- Python returns the value **None, if no `return` given**
- represents the absence of a value

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

`return` vs. `print`

- return only has meaning **inside** a function
- only **one** return executed inside a function
- code inside function but after return statement not executed
- has a value associated with it, **given to function caller**

- print can be used **outside** functions
- can execute **many** print statements inside a function
- code inside function can be executed after a print statement
- has a value associated with it, **outputted** to the console

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Exercise: Maximum of Two Numbers

- Write a function called **`find_maximum()`** that takes **two** numbers as arguments and returns the maximum of the two numbers. The function should compare the two numbers and determine which one is larger, and then return the larger number as the result.
- **`def find_maximum(num1, num2):`**

Iteration vs. Recursion

```
def factorial_iter(n):
    prod = 1
    for i in range(1,n+1):
        prod *= i
    return prod
```

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n*factorial(n-1)
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Class

- Object-Oriented Programming (OOP):
  - Python is an object-oriented programming language, which means it focuses on creating objects as the primary building blocks of programs.
  - OOP allows us to organize code into modular and reusable units, making it easier to manage and maintain complex projects.

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Class

- Class Definition:
  - A class is defined using the **`class`** keyword followed by the name of the class.
  - It serves as a blueprint that defines the properties and behaviors (methods) that objects of that class will possess.
  - For example, a class called **`Car`** can define properties like **`color`**, **`make`**, and **`model`**, as well as behaviors like **`start_engine()`** or **`accelerate()`**.

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Class

- Objects and Instances:
  - Once a class is defined, we can create objects or instances of that class.
  - An object is an instance of a class, representing a specific occurrence or entity.
  - For instance, we can create multiple instances of the **`Car`** class, each representing a different car with its own set of properties and behaviors.

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Class

In [1]:
```python
# Define a class called 'Person'
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        print(f"Hello, my name is {self.name} and I am {self.age} years old.")

# Create an instance of the 'Person' class
person1 = Person("Alice", 25)

# Access attributes of the person1 object
print(person1.name)  # Output: Alice
print(person1.age)   # Output: 25

# Call the greet() method of the person1 object
person1.greet()  # Output: Hello, my name is Alice and I am 25 years old.
```

```
Alice
25
Hello, my name is Alice and I am 25 years old.
```

Class

In [1]:
```python
# Define a class called 'Person'
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        print(f"Hello, my name is {self.name} and I am {self.age} years old.")

# Create an instance of the 'Person' class
person1 = Person("Alice", 25)

# Access attributes of the person1 object
print(person1.name)  # Output: Alice
print(person1.age)   # Output: 25

# Call the greet() method of the person1 object
person1.greet()  # Output: Hello, my name is Alice and I am 25 years old.
```

```
Alice
25
Hello, my name is Alice and I am 25 years old.
```

Class

In [1]:
```python
# Define a class called 'Person'
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        print(f"Hello, my name is {self.name} and I am {self.age} years old.")

# Create an instance of the 'Person' class
person1 = Person("Alice", 25)

# Access attributes of the person1 object
print(person1.name)  # Output: Alice
print(person1.age)   # Output: 25

# Call the greet() method of the person1 object
person1.greet()  # Output: Hello, my name is Alice and I am 25 years old.
```

```
Alice
25
Hello, my name is Alice and I am 25 years old.
```

Exercise: Bank Account Class

- Define a class called **`BankAccount`**.
- The class should have the following attributes: **`account_number`**, **`owner_name`**, and **`balance`**.
- Implement the **`__init__()`** method to initialize the attributes.
- Implement the following methods:
  - **`deposit(amount)`**: Adds the given amount to the account's balance.
  - **`withdraw(amount)`**: Subtracts the given amount from the account's balance.
  - **`get_balance()`**: Returns the current balance of the account.
  - **`display_account_info()`**: Prints the account information (account number, owner name, and balance).

Exercise: Bank Account Class

```python
# Define the BankAccount class
class BankAccount:
    def __init__(self, account_number, owner_name, balance=0):
        self... = ...

    def deposit(self, amount):
        # Adds the given amount to the account's balance.

    def withdraw(self, amount):
        # Subtracts the given amount from the account's balance.

    def get_balance(self):
        # Returns the current balance of the account.

    def display_account_info(self):
        # Prints the account information
        # (account number, owner name, and balance).


# Create an instance of the BankAccount class
account1 = BankAccount("123456789", "Alice", 1000)
```

Tuples

- an ordered sequence of elements, can mix element types
- cannot change element values, **immutable**
- represented with parentheses

remember strings?

```
te = ()
```
empty tuple

```
t = (2,"mit",3)
t[0]                    → evaluates to 2
(2,"mit",3) + (5,6)     → evaluates to (2,"mit",3,5,6)
t[1:2]      → slice tuple, evaluates to ("mit",)
t[1:3]      → slice tuple, evaluates to ("mit",3)
len(t)      → evaluates to 3
t[1] = 4    → gives error, can't modify object
```

extra comma means a tuple with one element

Tuples

- conveniently used to **swap** variable values

```
x = y
y = x
```

```
temp = x
x = y
y = temp
```

```
(x, y) = (y, x)
```

- used to **return more than one value** from a function

```
def quotient_and_remainder(x, y):
    q = x // y
    r = x % y
    return (q, r)
(quot, rem) = quotient_and_remainder(4,5)
```

integer division

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Tuples

- Iterate over tuples

```python
# Define a tuple of fruits
fruits = ('apple', 'banana', 'orange', 'mango')

# Iterate over the elements of the tuple
for fruit in fruits:
    print(fruit)

# Output:
# apple
# banana
# orange
# mango
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Lists

- **ordered sequence** of information, accessible by index
- a list is denoted by **square brackets**, [ ]
- a list contains **elements**
  - usually homogeneous (ie, all integers)
  - can contain mixed types (not common)
- list elements can be changed so a list is **mutable**

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Indices and Ordering

```
a_list = []
```

empty list

```
L = [2, 'a', 4, [1,2]]
len(L)  → evaluates to 4
L[0]    → evaluates to 2
L[2]+1  → evaluates to 5
L[3]    → evaluates to [1,2], another list!
L[4]    → gives an error
i = 2
L[i-1]  → evaluates to 'a' since L[1]='a' above
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Iterating Over a List

- compute the **sum of elements** of a list
- common pattern, iterate over list elements

like strings, can iterate over list elements directly

```
total = 0
for i in range(len(L)):
    total += L[i]
print total
```

```
total = 0
for i in L:
    total += i
print total
```

- notice
  - list elements are indexed `0` to `len(L)-1`
  - `range(n)` goes from `0` to `n-1`

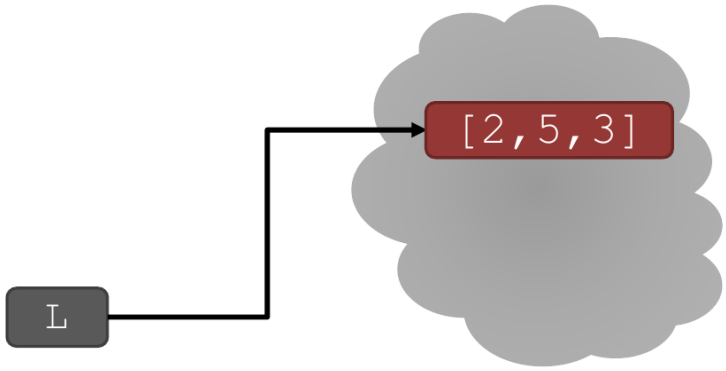
Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Changing Elements

- lists are **mutable**!
- assigning to an element at an index changes the value

```
L = [2, 1, 3]
L[1] = 5
```

- `L` is now `[2, 5, 3]`, note this is the **same object** `L`



Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Operations on Lists: Add

- **add** elements to end of list with `L.append(element)`
- **mutates** the list!

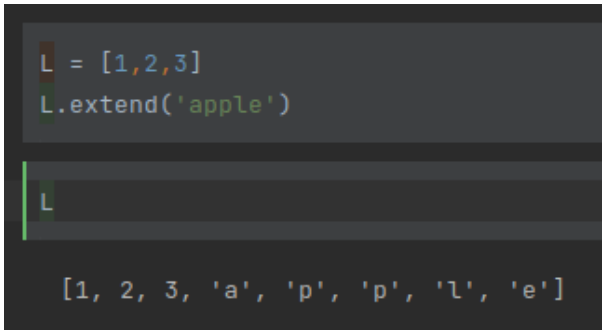
```
L = [2,1,3]
L.append(5)
```

→ L is now `[2,1,3,5]`

- what is the dot?
  - lists are Python objects, everything in Python is an object
  - objects have data
  - objects have methods and functions
  - access this information by `object_name.do_something()`
  - will learn more about these later

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Operations on Lists: Add



- to combine lists together use **concatenation**, + operator, to give you a new list
- **mutate** list with `L.extend(some_list)`

```
L1 = [2,1,3]
L2 = [4,5,6]
L3 = L1 + L2
```

→ `L3` is `[2,1,3,4,5,6]`
`L1`, `L2` unchanged

```
L1.extend([0,6])
```

→ mutated `L1` to `[2,1,3,0,6]`

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Operations on Lists: Remove

- delete element at a **specific index** with `del(L[index])`
- remove element at **end of list** with `L.pop()`, returns the removed element
- remove a **specific element** with `L.remove(element)`
  - looks for the element and removes it
  - if element occurs multiple times, removes first occurrence
  - if element not in list, gives an error

all these operations mutate the list

```
L = [2,1,3,6,3,7,0] # do below in order
L.remove(2) → mutates L = [1,3,6,3,7,0]
L.remove(3) → mutates L = [1,6,3,7,0]
del(L[1])   → mutates L = [1,3,7,0]
L.pop()     → returns 0 and mutates L = [1,3,7]
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Exercise: Tuple and List

- Create a tuple named **`my_tuple`** with the following elements: "apple", "banana", "cherry", "apple".
- Create a list named **`my_list`** with the same elements as **`my_tuple`**.
  - Hint: you can use **`list(my_tuple)`**
- Print the length of **`my_tuple`**.
- Print the length of **`my_list`**.
- Add an element "orange" to **`my_list`** using the appropriate method.
- Remove the element "cherry" from **`my_list`** using the appropriate method.
- Print **`my_list`** to verify the modifications.
- Use the **`pop()`** method to remove and print the last element from **`my_list`**.
- Print the modified **`my_list`** to verify the modifications.

Convert Lists to Srtings

- convert **string to list** with `list(s)`, returns a list with every character from `s` an element in `L`
- can use `s.split()`, to **split a string on a character** parameter, splits on spaces if called without a parameter
- use `''.join(L)` to turn a **list of characters into a string**, can give a character in quotes to add char between every element

```
s = "I<3 cs"          → s is a string
list(s)               → returns ['I','<','3',' ','c','s']
s.split('<')          → returns ['I', '3 cs']
L = ['a','b','c']     → L is a list
''.join(L)            → returns "abc"
'_'.join(L)           → returns "a_b_c"
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Other List Operations

- `sort()` and `sorted()`
- `reverse()`
- and many more!
  https://docs.python.org/3/tutorial/datastructures.html

```
L=[9,6,0,3]
```

`sorted(L)` → returns sorted list, does **not mutate** `L`

`L.sort()` → **mutates** `L=[0,3,6,9]`

`L.reverse()` → **mutates** `L=[9,6,3,0]`

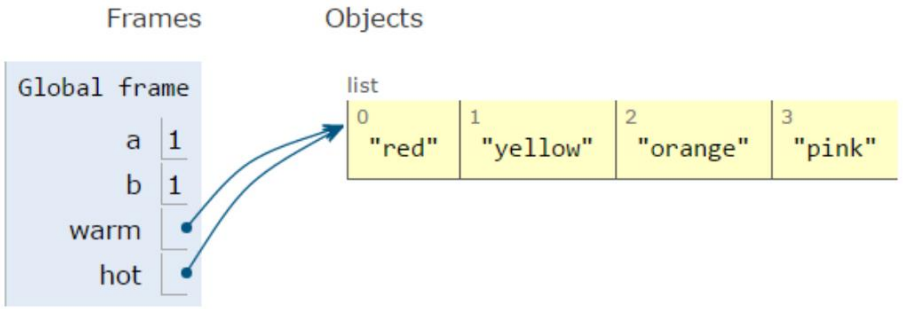
Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Aliases

- `hot` is an **alias** for `warm` – changing one changes the other!
- `append()` has a side effect

```
1  a = 1
2  b = a
3  print(a)
4  print(b)
5
6  warm = ['red', 'yellow', 'orange']
7  hot = warm
8  hot.append('pink')
9  print(hot)
10 print(warm)
```

```
1
1
['red', 'yellow', 'orange', 'pink']
['red', 'yellow', 'orange', 'pink']
```



Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Cloning a List

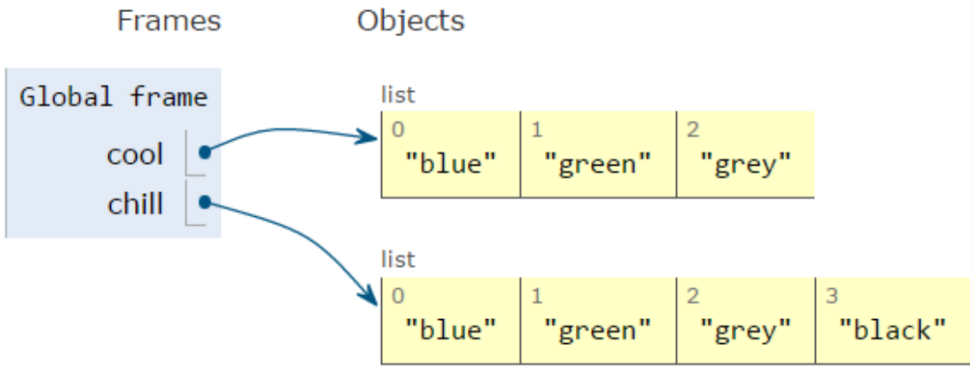
```
L_1 = [1,2,3]
L_2 = L_1.copy()
L_1.append(4)
print(L_1)
print(L_2)
```

```
[1, 2, 3, 4]
[1, 2, 3]
```

- create a new list and **copy every element** using `chill = cool[:]`

```
1 cool = ['blue', 'green', 'grey']
2 chill = cool[:]
3 chill.append('black')
4 print(chill)
5 print(cool)
```

```
['blue', 'green', 'grey', 'black']
['blue', 'green', 'grey']
```



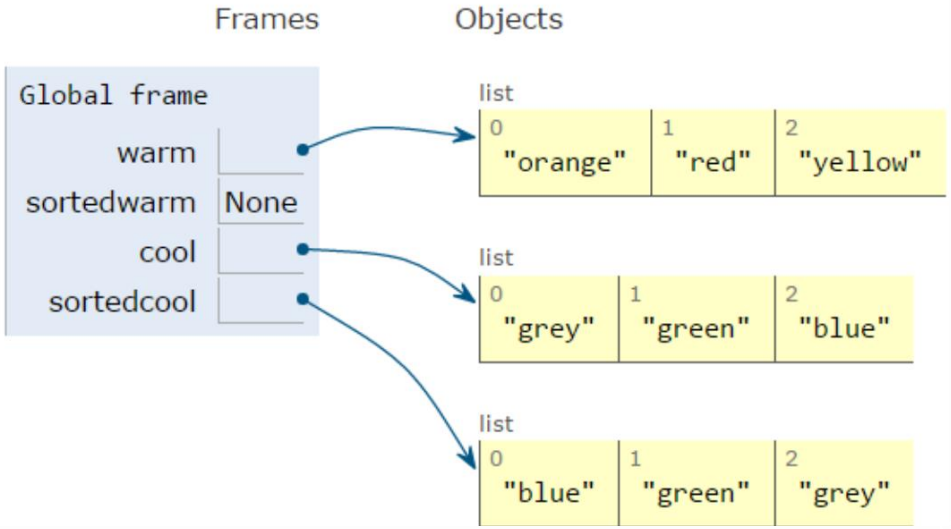
Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Cloning a List

- calling `sort()` **mutates** the list, returns nothing
- calling `sorted()` **does not mutate** list, must assign result to a variable

```
['orange', 'red', 'yellow']
None
['grey', 'green', 'blue']
['blue', 'green', 'grey']
```

```
1 warm = ['red', 'yellow', 'orange']
2 sortedwarm = warm.sort()
3 print(warm)
4 print(sortedwarm)
5
6 cool = ['grey', 'green', 'blue']
7 sortedcool = sorted(cool)
8 print(cool)
9 print(sortedcool)
```

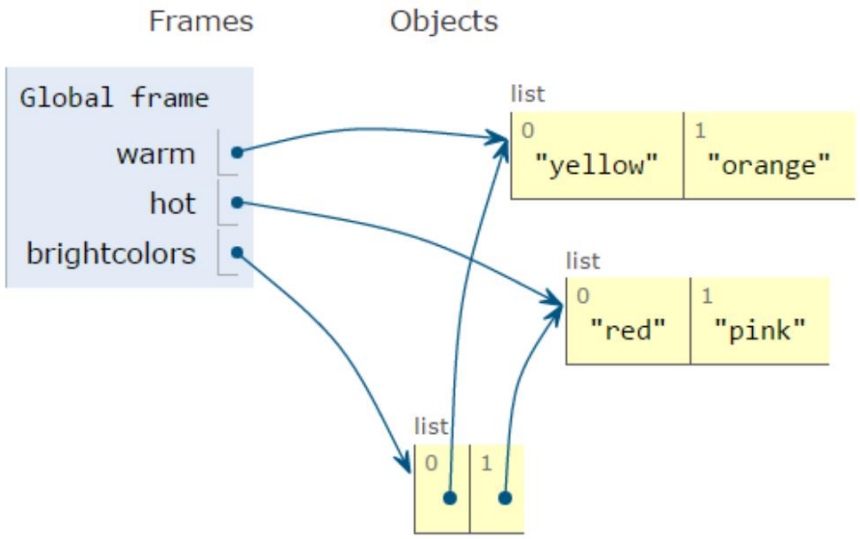


Nested Lists

- can have **nested** lists
- side effects still possible after mutation

```
[['yellow', 'orange'], ['red']]
['red', 'pink']
[['yellow', 'orange'], ['red', 'pink']]
```

```
1 warm = ['yellow', 'orange']
2 hot = ['red']
3 brightcolors = [warm]
4 brightcolors.append(hot)
5 print(brightcolors)
6 hot.append('pink')
7 print(hot)
8 print(brightcolors)
```



Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Avoiding mutation during iteration

- **avoid** mutating a list as you are iterating over it

```
def remove_dups(L1, L2):
    for e in L1:
        if e in L2:
            L1.remove(e)
```

```
def remove_dups(L1, L2):
    L1_copy = L1[:]
    for e in L1_copy:
        if e in L2:
            L1.remove(e)
```

```
L1 = [1, 2, 3, 4]
L2 = [1, 2, 5, 6]
remove_dups(L1, L2)
```

clone list first, note that `L1_copy = L1` does NOT clone

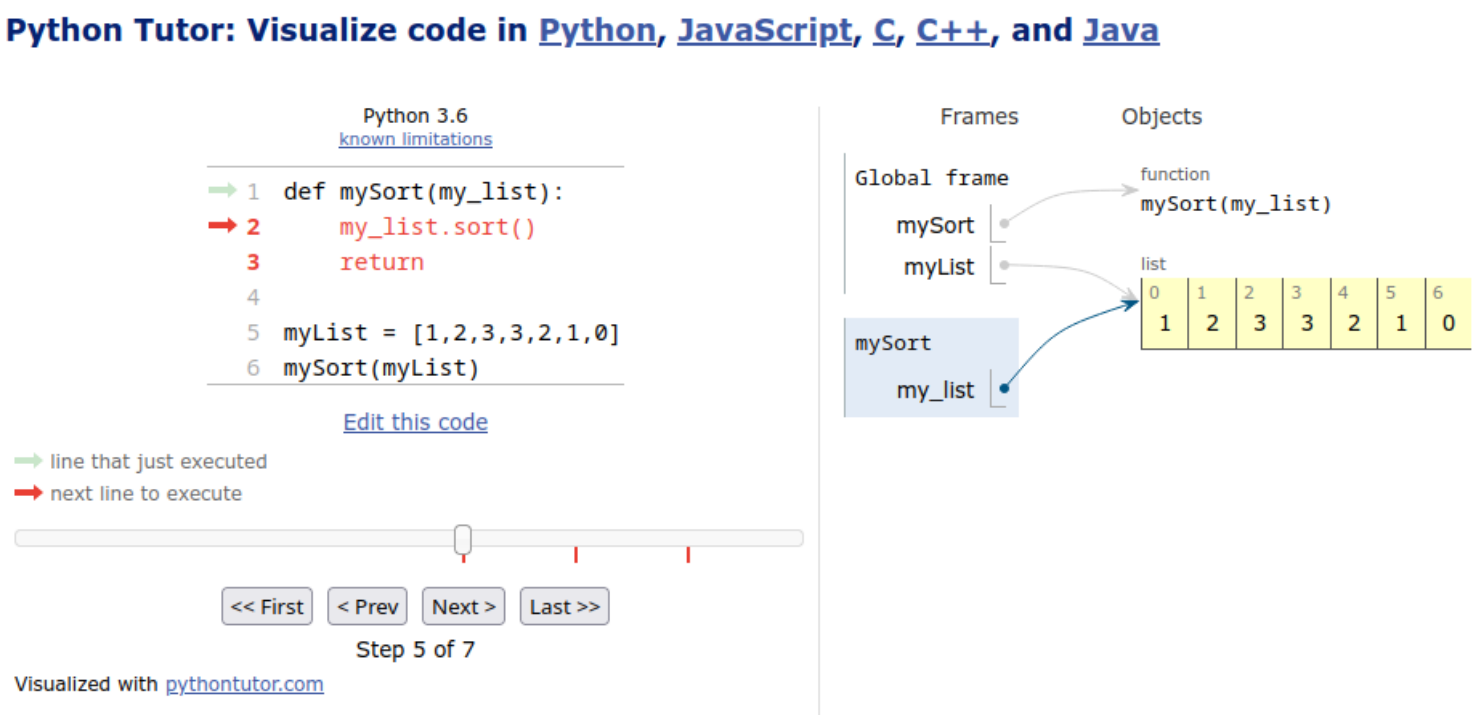
- `L1` is `[2,3,4]` not `[3,4]` Why?
  - Python uses an internal counter to keep track of index it is in the loop
  - mutating changes the list length but Python doesn't update the counter
  - loop never sees element 2

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

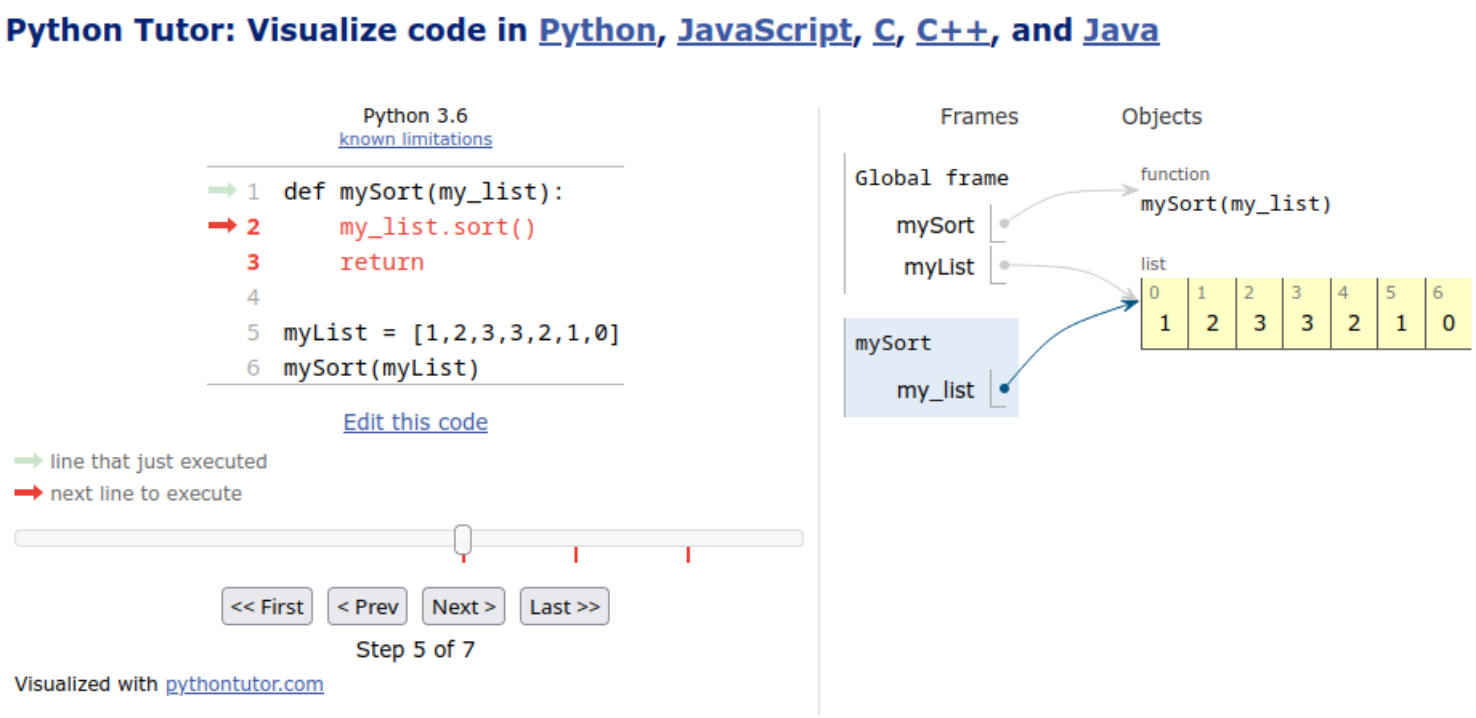
Exercise: List Operations Function

1. Write a function called **`list_operations`** that takes a list as an argument.
2. Within the function, perform the following operations on the list:
   - **Clone the original list** and assign it to a new variable.
   - Sort the original list in ascending order.
   - **Create an alias of the original list** by assigning it to a new variable.
   - Reverse the alias of the list
3. Return the cloned list, sorted list, and the alias of the original list.
4. Test the function by calling it with different lists and printing the results.

Exercise: List Operations Function



Exercise: List Operations Function



- Dictionary
  - e.g., how to store student info

    - so far, can store using separate lists for every info

```
names = ['Ana', 'John', 'Denise', 'Katy']
grade = ['B', 'A+', 'A', 'A']
course = [2.00, 6.0001, 20.002, 9.01]
```

    - a **separate list** for each item
    - each list must have the **same length**
    - info stored across lists at **same index**, each index refers to info for a different person

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

- Dictionary
  - e.g., how to store student info

```
def get_grade(student, name_list, grade_list, course_list):
    i = name_list.index(student)
    grade = grade_list[i]
    course = course_list[i]
    return (course, grade)
```

- **messy** if have a lot of different info to keep track of
- must maintain **many lists** and pass them as arguments
- must **always index** using integers
- must remember to change multiple lists

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

- Dictionary
  - e.g., how to change student info
  - dict: a better and clearner way
  - nice to **index item of interest directly** (not always int)
  - nice to use **one data structure**, no separate lists

A list

0	Elem 1
1	Elem 2
2	Elem 3
3	Elem 4
…	…

index element

A dictionary

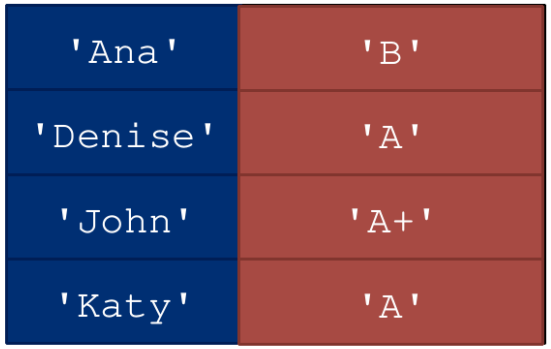
Key 1	Val 1
Key 2	Val 2
Key 3	Val 3
Key 4	Val 4
…	…

custom index by label element

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Python Dictionary

- store pairs of data
  - key
  - value



empty dictionary

```
my_dict = {}
```

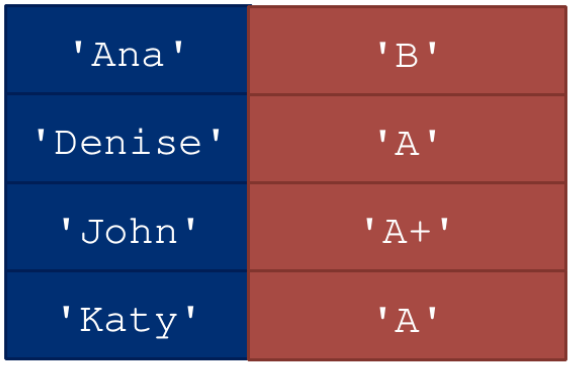
```
grades = {'Ana':'B', 'John':'A+', 'Denise':'A', 'Katy':'A'}
```

key1 val1 key2 val2 key3 val3 key4 val4

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Dictionary Lookup

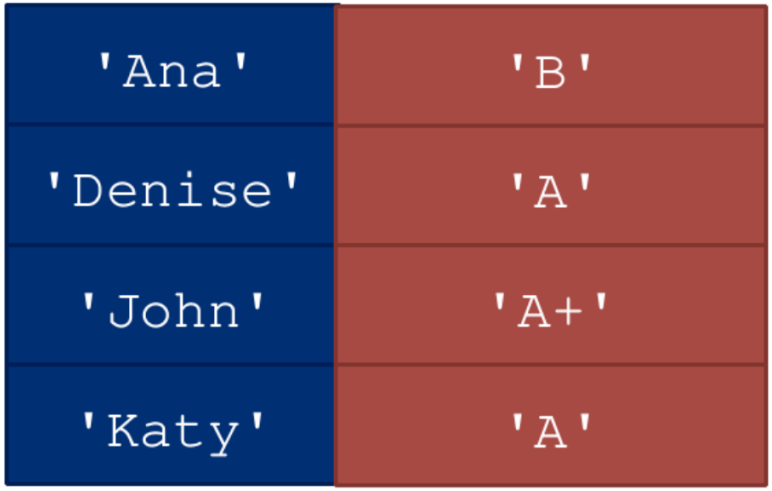
- similar to indexing into a list
- **looks up** the **key**
- **returns** the **value** associated with the key
- if key isn't found, get an error




```
grades = {'Ana':'B', 'John':'A+', 'Denise':'A', 'Katy':'A'}
grades['John']      → evaluates to 'A+'
grades['Sylvan']    → gives a KeyError
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Dictionary Operations




```
grades = {'Ana':'B', 'John':'A+', 'Denise':'A', 'Katy':'A'}
```

- **add** an entry

```
grades['Sylvan'] = 'A'
```

- **test** if key in dictionary

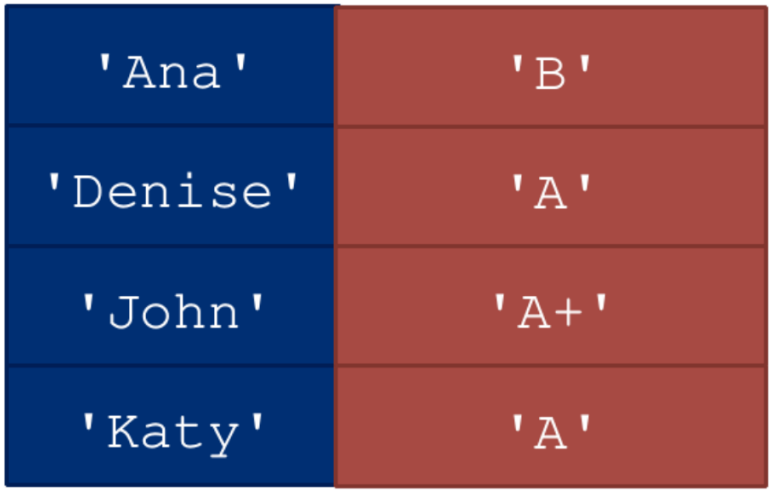
```
'John' in grades        → returns True
'Daniel' in grades      → returns False
```

- **delete** entry

```
del(grades['Ana'])
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Dictionary Operations




```
grades = {'Ana':'B', 'John':'A+', 'Denise':'A', 'Katy':'A'}
```

- get an **iterable that acts like a tuple of all keys** (no guaranteed order)

  `grades.keys()` → returns `['Denise','Katy','John','Ana']`

- get an **iterable that acts like a tuple of all values**

  `grades.values()` → returns `['A', 'A', 'A+', 'B']` (no guaranteed order)

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Dictionary Keys and Values

```python
students = {
    "John": {"grade": 85, "course": "Math"},
    "Jane": {"grade": 92, "course": "Science"},
    "Michael": {"grade": 78, "course": "English"}
}
```

- values
  - any type (**immutable and mutable**)
  - can be **duplicates**
  - dictionary values can be lists, even other dictionaries!
- keys
  - must be **unique**
  - **immutable** type (`int`, `float`, `string`, `tuple`, `bool`)
    - actually need an object that is **hashable,** but think of as immutable as all immutable types are hashable
  - careful with `float` type as a key
- **no order** to keys or values!

```
d = {4:{1:0}, (1,3):"twelve", 'const':[3.14,2.7,8.44]}
```

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

`list` vs `dict`

- **ordered** sequence of elements
- look up elements by an integer index
- indices have an **order**
- index is an **integer**

- **matches** "keys" to "values"
- look up one item by another item
- **no order** is guaranteed
- key can be any **immutable** type

Credit: Ana Bell, Eric Grimson, John Guttag (2016). *Introduction to Computer Science and Programming in Python* [PowerPoint slides].

Exercise: Student Database

- Create an empty dictionary called **`student_db`**.
- Add the following student records to the **`student_db`** dictionary; using student id as the key:
  - Student ID: 101, Name: Alice, Grade: A
  - Student ID: 102, Name: Bob, Grade: B
  - Student ID: 103, Name: Charlie, Grade: C

```
student_db = {}
student_db[some_id] = {"Name": "some_name", "Grade": "some_grade"}
```

- Print the student records in the following format:
- Update the grade of student with ID 102 to "A+".
- Print the updated student record of student with ID 102.
- Check if student with ID 104 exists in the **`student_db`** dictionary.
- Print "Student ID 104 exists" if it exists, or "Student ID 104 does not exist" if it doesn't.
- Remove the student record of student with ID 103 from the **`student_db`** dictionary.
- Print the remaining student records after the removal in the same format as step 3.
- Clear all records from the **`student_db`** dictionary.
- Print the dictionary to confirm that it is empty.