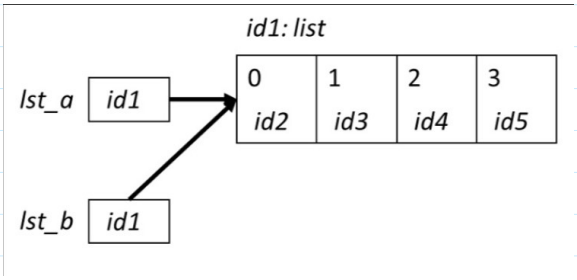
□ Aliasing

Tuesday, 15 March 2022 20:50

1. Aliasing

Definition: When two variable names refer to the same object, they are aliases. When we modifying one variable, we are modifying the object it refers to, hence modifying second value.



⊛ Aliasing can happen in all mutable objects (lists, sets, dictionaries)

⇒ Whenever we use object1 = object2, and they refer to mutable.

2. Aliasing and Function Calls

Recall: Any variable defined inside a function is only within local scope.
Any variable defined outside will cover within global scope

- **Immutable Type** When we pass immutable to a function, for example, x, the value of x is not changed after function. Note that we have to use function (x) to get the value modified by the function

- **Mutable Type:** When we pass a list to a function, the function gets a reference to the list

→ If the function modifies the list parameter, that change IS REFLECTED AT GLOBAL SCOPE.

(reason why we use list.append(), not list = list.append())

In [8]:
```
def g(y):                    ⇒ still refering to old list
    y = ['a','b', 'c']       ⇒ reassigned y
    return y

y = [1,2,3]
y_new = g(y)
print("y =", y, "\ty_new =", y_new)
```

y = [1, 2, 3] y_new = ['a', 'b', 'c']

In [9]:
```
def g(y):
    y[0] = ['a','b', 'c']
    return y

y = [1,2,3]
y_new = g(y)
print("y =", y, "\ty_new =", y_new)
```

y = [['a', 'b', 'c'], 2, 3] y_new = [['a', 'b', 'c'], 2, 3]

Default Function Values

Tuesday, 15 March 2022 21:43

Are values that will be used if no parameter is passed.

e.g. For print(), default values are:

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

NOTE If we want to change end=, REMEMBER TO ADD '\n'
(If we still want to go to next line on next one)

- We can set default values for functions. If not overwritten, they will be used as parameters.

```
def make_greeting(title, name, surname, formal=True):
  if formal:
    return "Hello " + title + " " + surname

  return "Hello " + name

print(make_greeting("Mr.", "John", "Smith"))
print(make_greeting("Mr.", "John", "Smith", False))
```

```
Hello Mr. Smith
Hello John
```

Friday, 18 March 2022 20:30

File Writing

Thursday, 17 March 2022 21:48

While program is running, its data is stored in RAM
⇒ Not saved after ending program
⇒ Need to save on files

Working with Files

+) A file must be opened
+) When done, it has to be closed
+) When file is open, it can either be read from or written to
+) The file keeps track of where you are reading to or writing from
+) You can read the whole file in its natural order or you can skip around

Opening a file

`open(filename, mode)`

- filename → external storage location
- mode → 3 modes

Writing to a File

`myfile = open('test.txt', 'w')`

- If there is no file named 'test.txt', it will be created
- If there is already one, it will be replaced by the new file.

→ `myfile = open('text.txt', 'a')` to append

- To write something we need to use write method : `myfile.write('...')`

NOTE The write method does not attach a new line character by default.
→ need ('\n')
→ Everytime we use myfile.write() a string is added to file where we left off

Closing a file

`myfile.close()`

- Need to close file after finishing to save what we did.
- This tells system that we are done writing and makes the disk file available for reading and writing by other programs (or by our program)

e.g. Store a dictionary into a file

```
students = {'Tina': 'A+', 'Brad': 'C-', 'Sue': 'B'}
# create a file
myfile = open("grades.txt", "w")
# store dictionary items to file
for student in students:
    myfile.write(student + ',' + students[student] + '\n')
# close the file
myfile.close()
```

File Reading

Thursday, 17 March 2022 23:11

```
myfile = open('text.txt', 'r')
```

If file doesn't exist → ERROR

Approach	Code	When to use it
The read approach	myfile = open(filename, 'r') contents = myfile.read() myfile.close()	When you want to read the whole file at once and use it as a single string.

The readline approach	myfile = open(filename, 'r')
	contents = ''
	line = myfile.readline()
	while line != '': → while line still exist
	  contents += line
	  line = myfile.readline()
	myfile.close()
		When you want to process only part of a file. Each time through the loop line contains one line of the file.

The for line in file approach	myfile = open(filename, 'r') contents = '' for line in myfile: contents += line myfile.close()	When you want to process every line in the file one at a time.
The readlines approach	myfile = open(filename, 'r') lines = myfile.readlines() myfile.close()	When you want to examine each line of a file by index.

→ output single string

→ readline() will go over line by line
first call: first line
second call: second line
...

→ Access lines one at a time in for loops

→ each line is an index of a list

readlines

```
In [11]: # Approach: readlines
         # When to use it: When you want to process the file line-by-line with an index
         # Example code
         myfile = open(filename, 'r')
         lines = myfile.readlines() # lines is a list of strings. Each entry in lines is a line of the file
         myfile.close()
         print(lines)
```

```
['Kendrick,A+\n', 'Dre,C-\n', 'Snoop,B\n']
```

NOTE: Each line means a string that has form: 'abcde\n'
A new line / in for loops, is defined as having \n. As a result, be careful when using print(line) ⇒ There is \n in line, and \n in print.

with statement

Thursday, 17 March 2022 23:45

For every open, we must do a close. So there is a function called with that automatically closes the file when the end of with block is reached.

```
with open(filename, mode) as variable:
    body
```

In [12]:
```
def f(file):
    print(file.read())

with open('test.txt', 'r') as file:
    f(file)

print("The next line")
```

→ since we are using print(), there is \n at the end.

```
CATS!
I <3 my second sentence
hola
The next line
```

CSV Files

Thursday, 17 March 2022 23:53

Comma Separated Values files (a type of file that stores data)

→ Store values and datas (tables) as values with comma.
→ .csv file

Name	Test1	Test2	Final
Kendrick	100	50	29
Dre	76	32	33
Snoop	25	75	95

=

grades.csv

```
Name,Test1,Test2,Final
Kendrick,100,50,29
Dre,76,32,33
Snoop,25,75,95
```

Reading a CSV file :

We can open a CSV file using open / with as . Then, we can see the content as :

'Name, Test1, Test2, Final \n Kendrick, 100, 50, 29 \n Dre, 76, 32, 33 \n...'

Then, to read CSV files, we can easily use CSV module

→ csv.reader ( )
→ Output an iterable, where lines are lists (sublists) of elements.

e.g

```
import csv
csvfile = open ("grades.csv", "r")
grades_reader = csv.reader (csvfile)

row_num = 1
for row in grades_reader :
    print ('Row #', row_num , ':' , row )
    row_num += 1

csvfile.close ()
```

/ or using with open ()

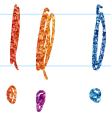
⊕ grades_reader is not recorded yet ⇒ need another for loop to record data to use

Row # 1 : ['Name', 'Test1', 'Test2', 'Final']
Row # 2 : ['Kendrick', '100', '50', '29']

Contain list of sublists.
Each sublists = 1 row.
⇒ .append (row) to keep rows as sublists.
If use += row to [],

Row # 1 : ['Name', 'Test1', 'Test2', 'Final']
Row # 2 : ['Kendrick', '100', '50', '29']
Row # 3 : ['Dre', '76', '32', '33']
Row # 4 : ['Snoop', '25', '75', '95']

rows as sublists.
If use += row to [], all rows will not be separated



→ Obtaining row & column information

Writing CSV Files

To write, we first would need to create a CSV writer object
`csv.writer()`

- Once object is created, we can use `writerow()` method to populate it with data.
  → writerow() can write a single row to file at a time

```
import csv

grades = [['Name', 'Test1', 'Test2', 'Final'],
          ['Kendrick', '100', '50', '69'],
          ['Dre', '76', '32', '53'],
          ['Snoop', '25', '75', '95']]

with open('grades_new.csv', 'w') as csvfile:
    grades_writer = csv.writer(csvfile)

    for row in grades:
        grades_writer.writerow(row)
```

Name	Test1	Test2	Final
Kendrick	100	50	69
Dre	76	32	53
Snoop	25	75	95

Opening CSV files in apps

In some cases, formatting error may occur, due to difference between newlines and carriage returns in Windows vs. Mac / Linux

To correct this error, we will need to prevent new line from forming → add parameter newline = ''

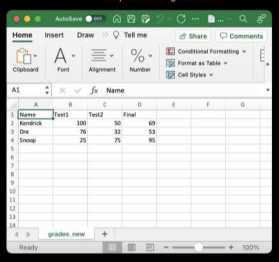
with open('grades_new.csv', 'w', newline='') as csvfile:

Summary: Reading + Writing

Friday, 18 March 2022 20:29

Summary: Reading Files

Requirement: "grades.csv"




```
##############################
# reading a file using standard approach
##############################

# open communication to a file (to read)
myfile = open('grades.csv', 'r')

# read from file
L = myfile.readlines()
for row in L:
    print(row, end = '')

myfile.close()
```

Requires find() method to obtain data!

```
##############################
# reading a file using CSV Module
##############################
import csv

# open communication to a file (to read)
myfile = open('grades.csv', 'r')

# read from file
csv_reader = csv.reader(myfile)
for row in csv_reader:
    print(row)

myfile.close()
```

Summary: Writing Files

Requirement: grades needs to be defined

```
grades = [['Name', 'Test1', 'Test2', 'Final'],
          ['Kendrick', '100', '50', '69'],
          ['Dre', '76', '32', '53'],
          ['Snoop', '25', '75', '95']]
```

```
##################################
#writing to a file using standard approach
##################################

# open communication to a file (to write)
myfile = open('new_grades.csv', 'w')

# write to file
for row in grades:
    for col in row:
        myfile.write(col + ',')
    myfile.write('\n')

myfile.close()
```

```
##############################
#writing to a file using CSV Module
##############################
import csv

# open communication to a file (to write)
myfile = open('new_grades.csv', 'w')

# write to file
grades_writer = csv.writer(myfile)
for row in grades:
    grades_writer.writerow(row)

myfile.close()
```

- with statement replaces open() & close
- read, write, append requires assigning / calling variable (if we still use open(), we still need to assign a variable to it)