INF1002
Programming Fundamentals
Lecture 3: Advanced Data Structure

Zhang Zhengchen

zhengchen.zhang@singaporetech.edu.sg

Review

- if elif elif else
  - Boolean expression
- while loop
- for loop
- f-string
  - Alignment
  - Width of a string
  - Precision of a float number

Outline

- Advanced Data Types
  - String
  - List
  - Tuple
  - Dictionary
- Files I/O

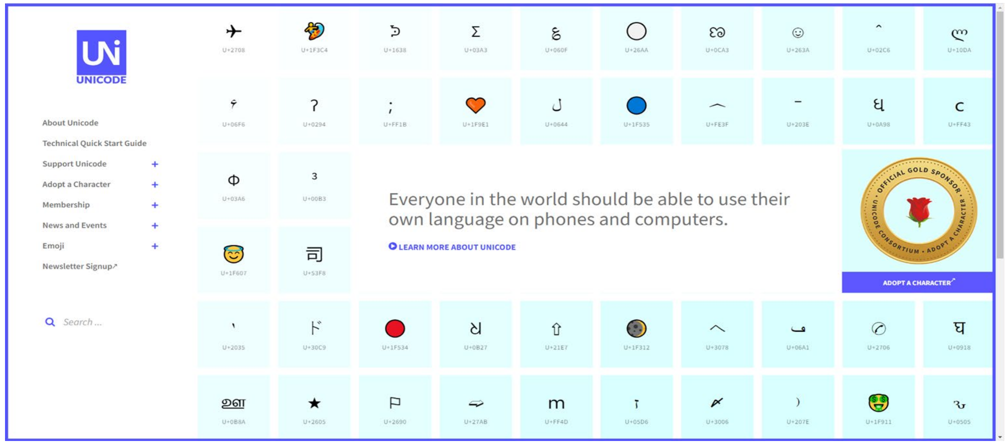
Type	Samples
int	8,12,1024
float	2.3, 3.1415926
bool	True, False
str	'Hello, World! ', '3.1415926'
None	None
List	
Tuple	
Set	
Dictionary	
byte	

string

- An immutable sequence of Unicode characters.
  - Immutable
  - Sequence
  - Unicode
- Immutable vs mutable
  - Mutable: The value of the object can be changed, such as lists, dictionaries, and sets.
  - Immutable: The value of the object cannot be changed once it is created, such as strings, tuples, and integers.

string

- An immutable sequence of Unicode characters.
    - Immutable
    - Sequence
    - Unicode
- Further Reading : Unicode
    - Unicode, formally The Unicode Standard is a text encoding standard maintained by the Unicode Consortium designed to support the use of text in all of the world's writing systems that can be digitized. [1]

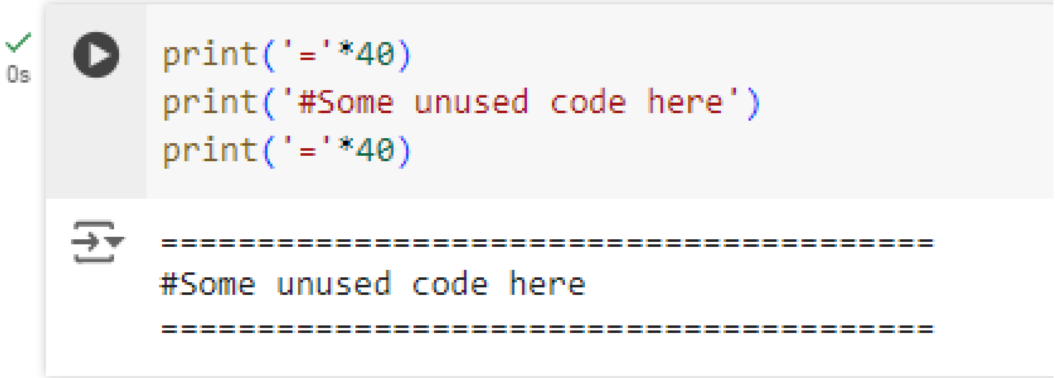


Everyone in the world should be able to use their own language on phones and computers.

1. https://en.wikipedia.org/wiki/Unicode
2. https://home.unicode.org/

string concatenation and repetition

- Strings are identified by single or double quotation marks:
  - 'Mike'
  - "Something interesting"
  - '73'
- The operator + is used for string concatenation:
  - 'Taylor' + 'Swift' evaluates to 'TaylorSwift'
  - 'Taylor' + ' '+ 'Swift' evaluates to 'Taylor Swift'
- Operator *

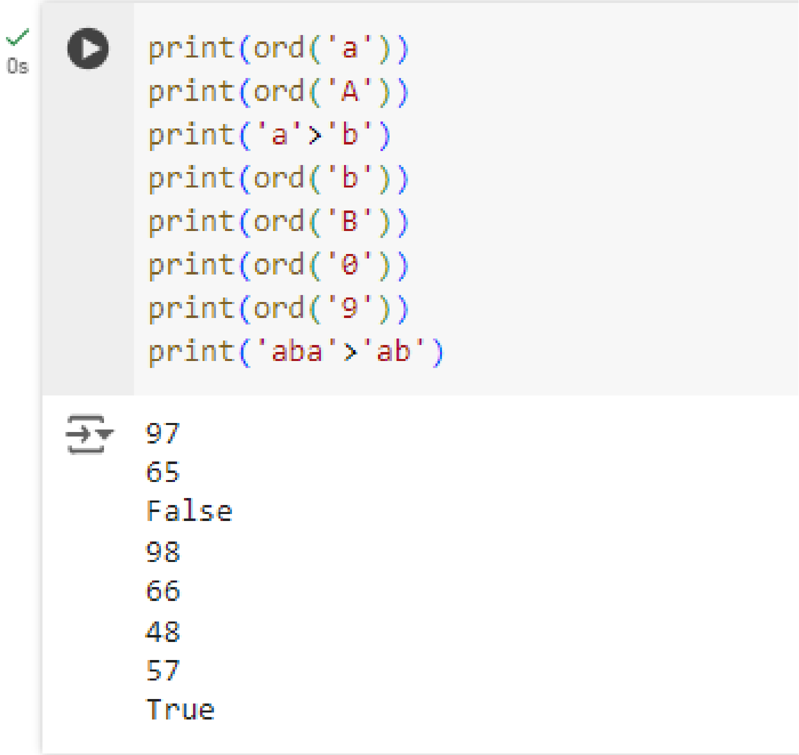


```
print('='*40)
print('#Some unused code here')
print('='*40)
```

```
========================================
#Some unused code here
========================================
```

string comparation

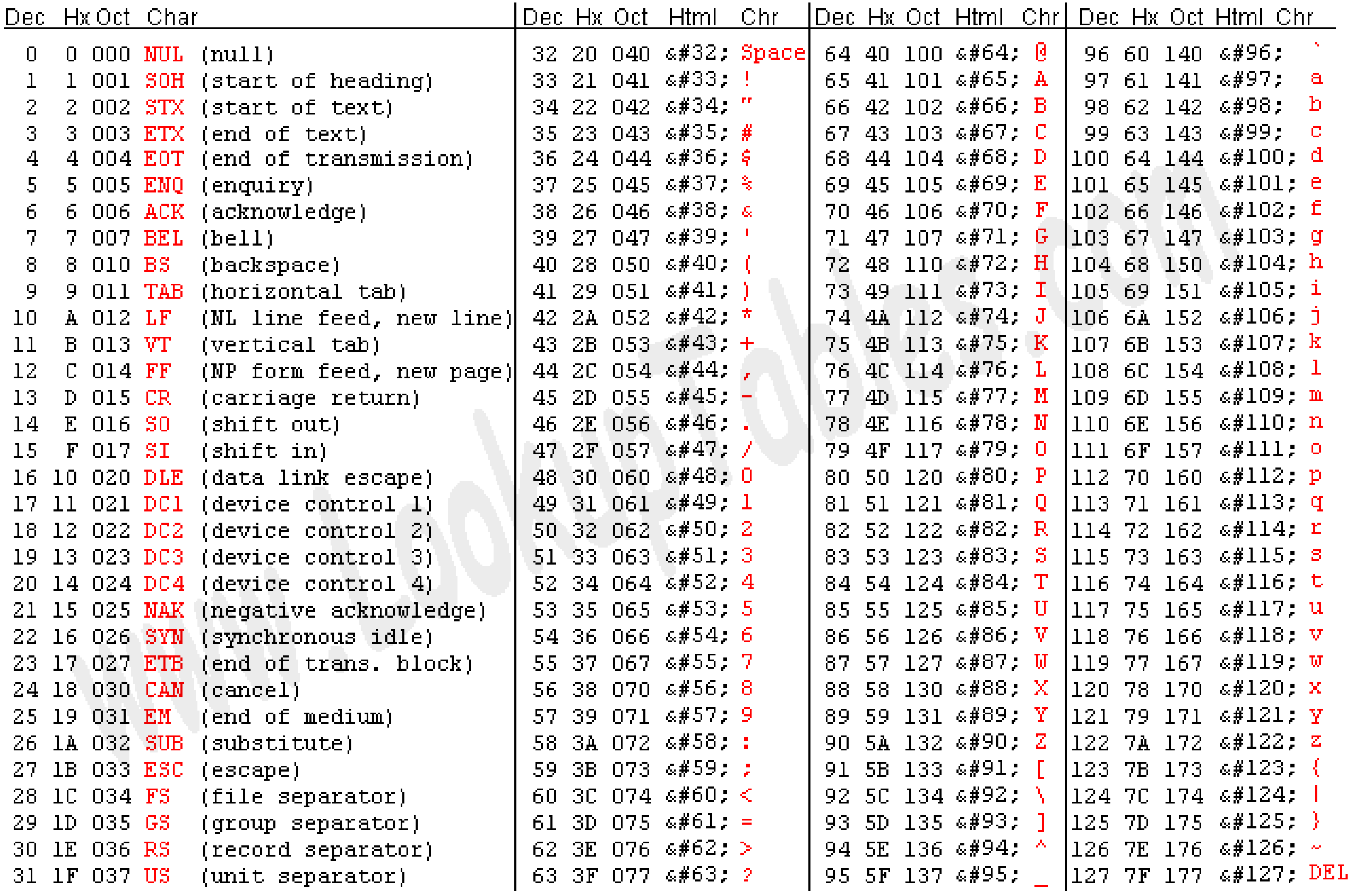
- What is the ASCII code of 'a' and 'b'
- Which is larger?
    - 'a' > 'b' or 'a' < 'b' ?
    - 'aa' > 'ab' or 'aa' < 'ab' ?
    - 'aba' > 'ab' or 'aba' < 'ab' ?
    - 'a0' > 'a9' or 'a0' < 'a9' ?



```
print(ord('a'))
print(ord('A'))
print('a'>'b')
print(ord('b'))
print(ord('B'))
print(ord('0'))
print(ord('9'))
print('aba'>'ab')
```

```
97
65
False
98
66
48
57
True
```

ASCII Table

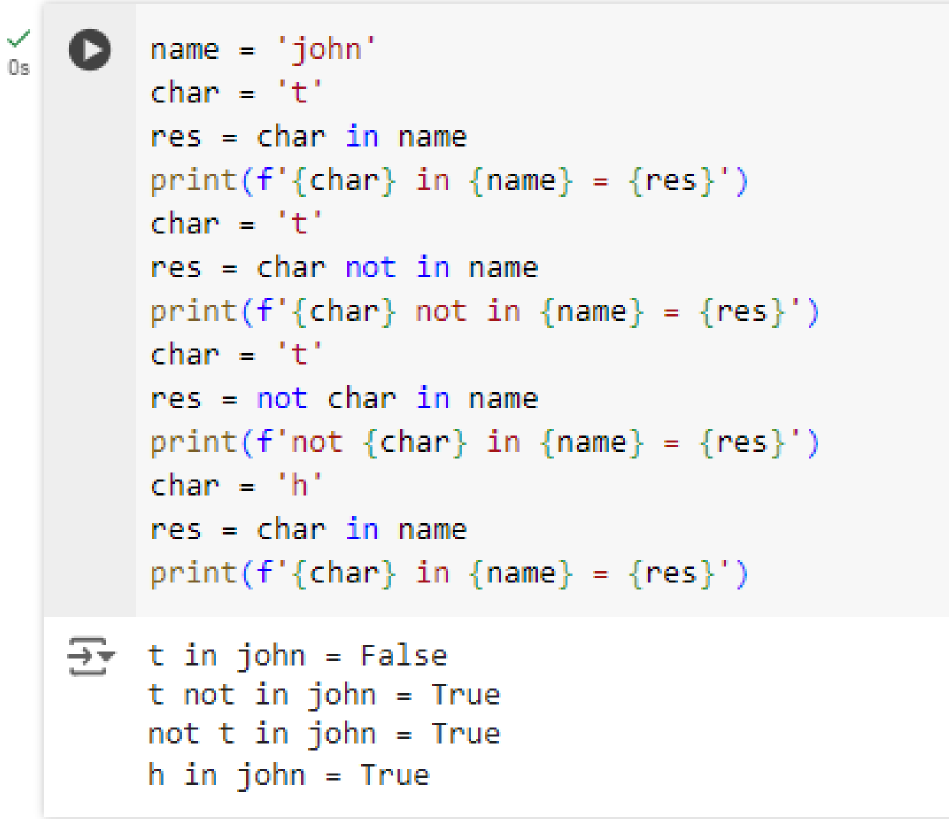


Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

string - a sequence of characters

- Sequence
- Membership Operator
  - in
  - not in

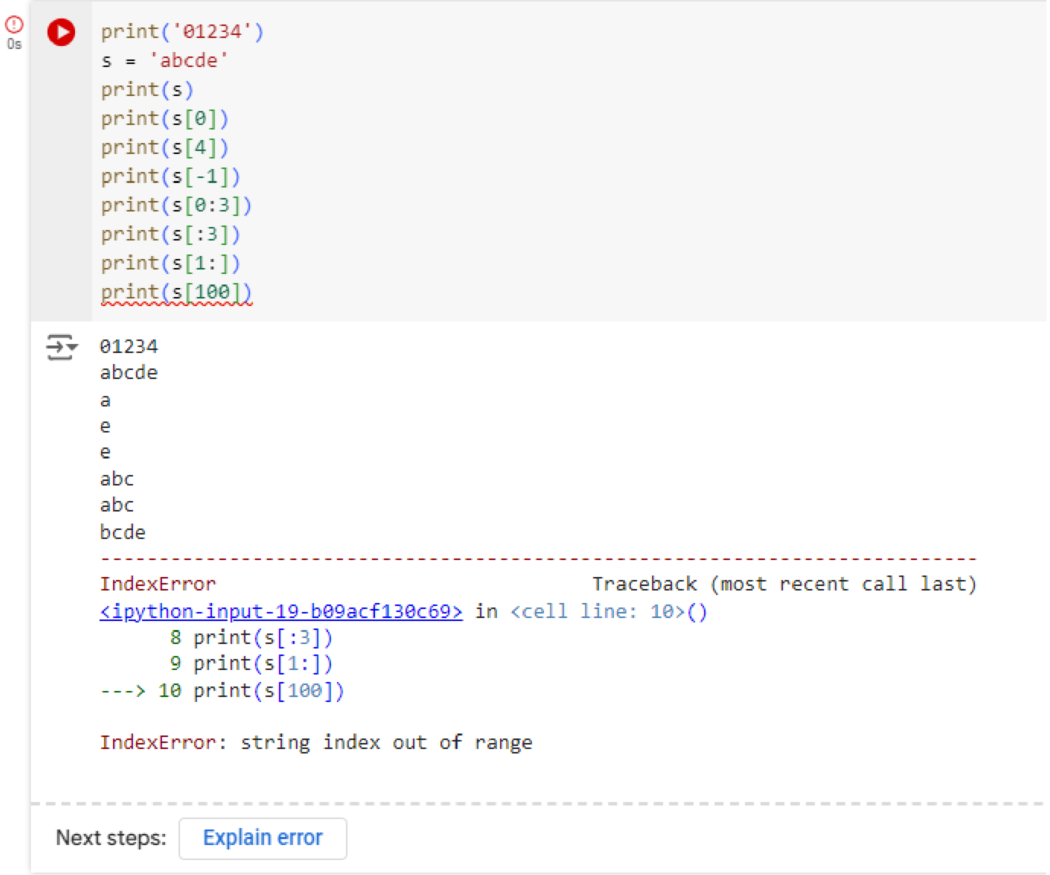


```python
name = 'john'
char = 't'
res = char in name
print(f'{char} in {name} = {res}')
char = 't'
res = char not in name
print(f'{char} not in {name} = {res}')
char = 't'
res = not char in name
print(f'not {char} in {name} = {res}')
char = 'h'
res = char in name
print(f'{char} in {name} = {res}')
```

```
t in john = False
t not in john = True
not t in john = True
h in john = True
```

string slicing

- Index of characters in a string
  - First character is indexed with 0.
  - Last character is indexed with len(s)-1
  - A character is a string with length 1
  - s[index]
  - s[-1]
  - s[start : end]
    - end: exclusive

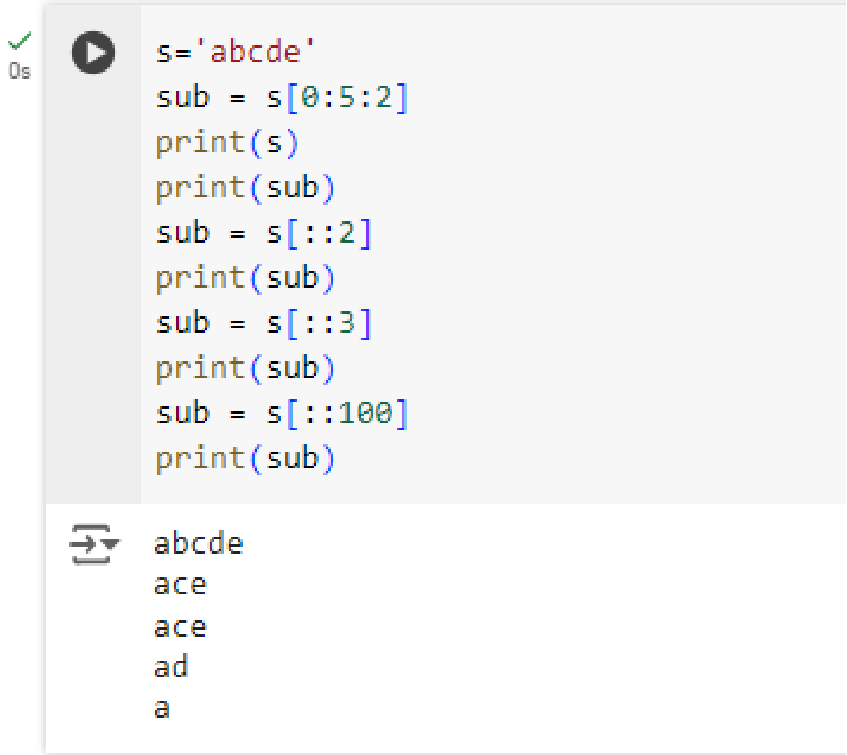


```
print('01234')
s = 'abcde'
print(s)
print(s[0])
print(s[4])
print(s[-1])
print(s[0:3])
print(s[:3])
print(s[1:])
print(s[100])
```

```
01234
abcde
a
e
e
abc
abc
bcde
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-19-b09acf130c69> in <cell line: 10>()
      8 print(s[:3])
      9 print(s[1:])
---> 10 print(s[100])

IndexError: string index out of range
```

string slicing

- Index of characters in a string
  - s[start : end : step]



```
s='abcde'
sub = s[0:5:2]
print(s)
print(sub)
sub = s[::2]
print(sub)
sub = s[::3]
print(sub)
sub = s[::100]
print(sub)
```

```
abcde
ace
ace
ad
a
```

string built-in methods

Function	Description
find(str, beg=0, end=len(string))	Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.
isdigit()	Returns true if string contains only digits and false otherwise.
lower()	Converts all uppercase letters in string to lowercase.
upper()	Converts lowercase letters in string to uppercase.
split(str="", num=string.count(str))	Splits string according to delimiter str (space if not provided) and returns list of substrings; split into at most num substrings if given.
endswith(suffix, beg=0, end=len(string))	Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise.
…	

String built-in functions

Name	Description
len(string)	Gives the total length of the string.
max(string)	Returns character from the string with max value.
min(string)	Returns character from the string with min value.

https://www.tutorialspoint.com/python/python_strings.htm

Function and Method

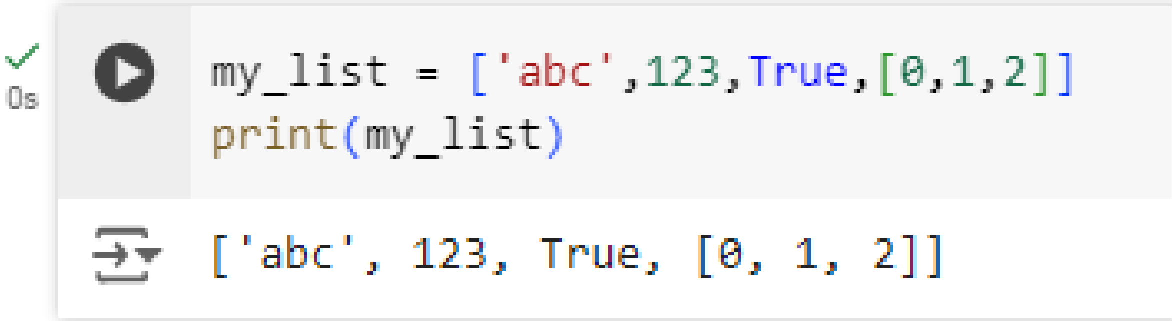
- Function
  - len(string)
- Method
  - string.lower()
- Further reading
  - Ask ChatGPT
  - Concept of function in python (we will learn this in next lecture)
  - Concept of Class/Object in Object-Oriented Programming

Review

- string
  - immutable and mutable
  - Concatenation and repetition
  - Comparison
  - String as a sequence
    - Membership operator
    - Slicing
  - Bulit-in functions and methods

List

- A Python list is a sequence of comma separated items, enclosed in square brackets [ ].
    - Name1 = 'Saul'
    - Name2 = 'David'
    - Name3 = 'Solomon'
    - Name4 = 'Rehoboam'
    - Names = ['Saul', 'David', 'Solomon', 'Rehoboam']
- The items in a Python list need not be of the same data type.

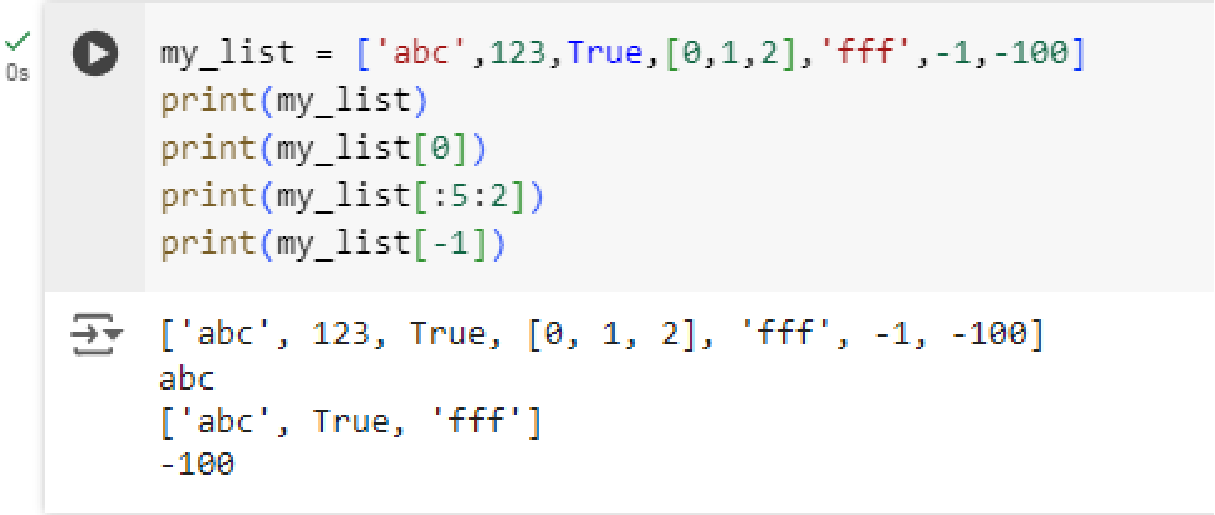


```
my_list = ['abc',123,True,[0,1,2]]
print(my_list)
```

```
['abc', 123, True, [0, 1, 2]]
```

List - Index in a sequence

- my_list[0]
- my_list[-1]
- Membership operator
  - 'abc' in my_list
- Slicing
  - my_list[start : end : step]

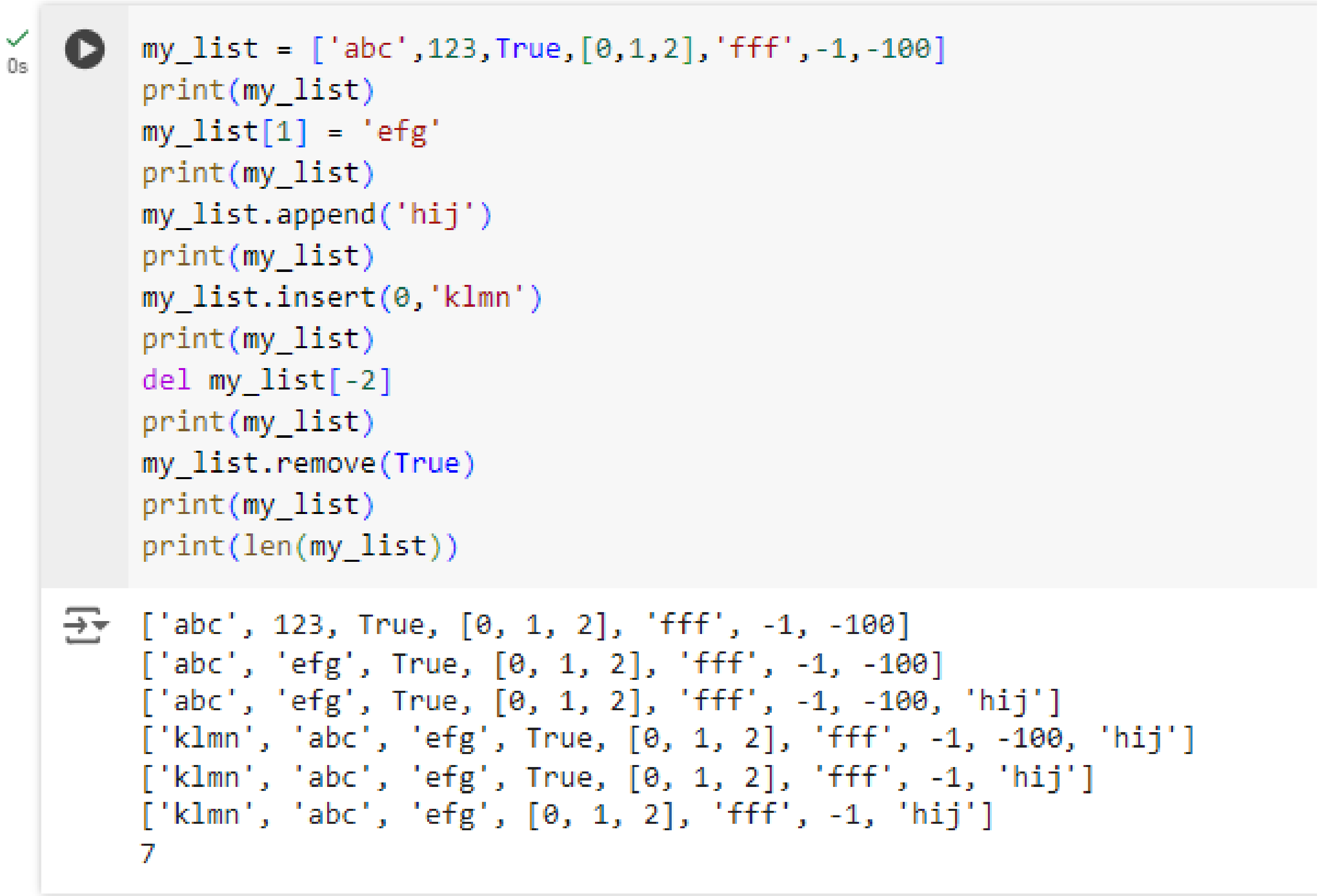


```
my_list = ['abc',123,True,[0,1,2],'fff',-1,-100]
print(my_list)
print(my_list[0])
print(my_list[:5:2])
print(my_list[-1])
```

```
['abc', 123, True, [0, 1, 2], 'fff', -1, -100]
abc
['abc', True, 'fff']
-100
```

List

- Update:
  - my_list[1] = 'efg'
  - Can string do this?
- Add
  - my_list.append('hij')
  - my_list.insert(0,'klmn')
- Delete
  - del my_list[-2]
  - my_list.remove('abc')
- Length
  - len(my_list)



```
my_list = ['abc',123,True,[0,1,2],'fff',-1,-100]
print(my_list)
my_list[1] = 'efg'
print(my_list)
my_list.append('hij')
print(my_list)
my_list.insert(0,'klmn')
print(my_list)
del my_list[-2]
print(my_list)
my_list.remove(True)
print(my_list)
print(len(my_list))
```

```
['abc', 123, True, [0, 1, 2], 'fff', -1, -100]
['abc', 'efg', True, [0, 1, 2], 'fff', -1, -100]
['abc', 'efg', True, [0, 1, 2], 'fff', -1, -100, 'hij']
['klmn', 'abc', 'efg', True, [0, 1, 2], 'fff', -1, -100, 'hij']
['klmn', 'abc', 'efg', True, [0, 1, 2], 'fff', -1, 'hij']
['klmn', 'abc', 'efg', [0, 1, 2], 'fff', -1, 'hij']
7
```

List

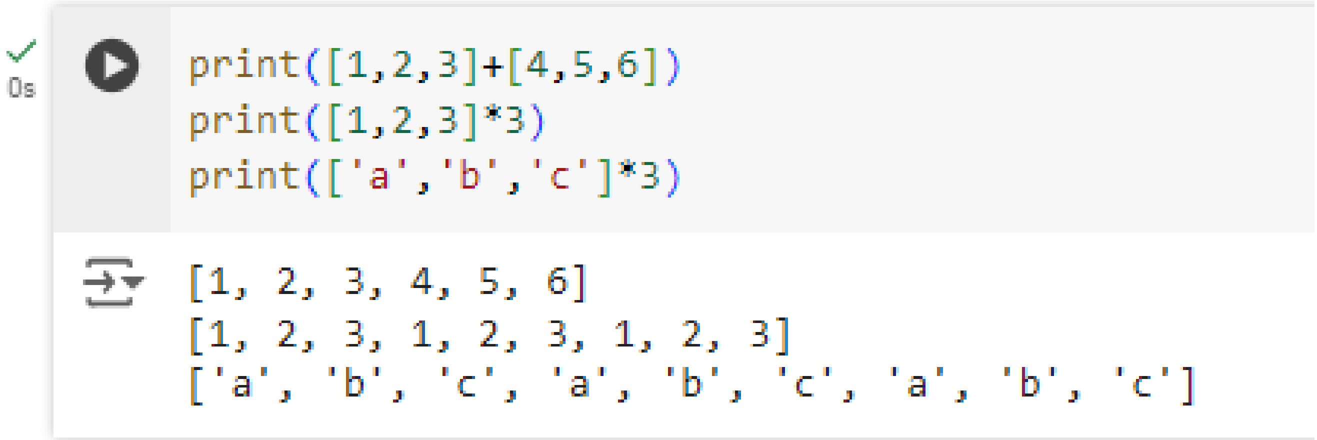
- Built-in functions

Name	Description
cmp(list1, list2)	Compares elements of both lists.
len(list)	Gives the total length of the list.
max(list)	Returns item from the list with max value.
min(list)	Returns item from the list with min value.
list(seq)	Converts a tuple into list.

https://www.tutorialspoint.com/python/python_lists.htm

List – Concatenation and Repetition

- Concatenation +
  - [1, 2, 3]+[4, 5, 6]
- Repetition *
  - [1, 2, 3]*4
  - ['a', 'b', 'c']*3

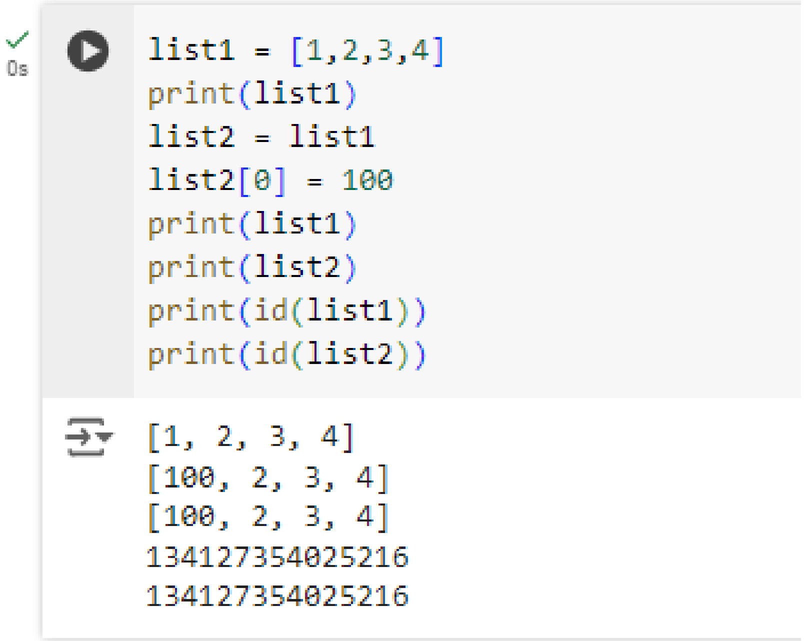


```
print([1,2,3]+[4,5,6])
print([1,2,3]*3)
print(['a','b','c']*3)
```

```
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 1, 2, 3, 1, 2, 3]
['a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c']
```

List

- Practice
  - Copy one list to a new list
  - Set the first item of the new list to be something else
  - Print the old and new lists

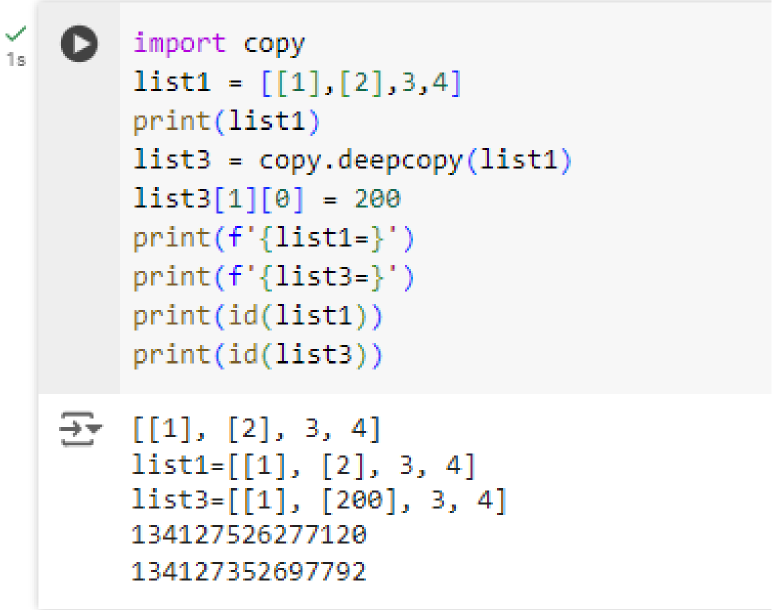


```
list1 = [1,2,3,4]
print(list1)
list2 = list1
list2[0] = 100
print(list1)
print(list2)
print(id(list1))
print(id(list2))
```

```
[1, 2, 3, 4]
[100, 2, 3, 4]
[100, 2, 3, 4]
134127354025216
134127354025216
```

Deep Copy

- A deep copy creates a new object and recursively adds copies of nested objects found in the original.
- Changes to the copied object do not affect the original object.

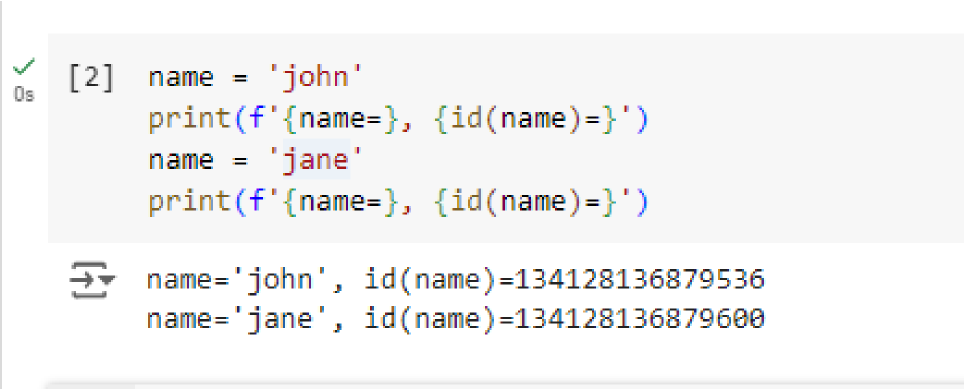


```
import copy
list1 = [[1],[2],3,4]
print(list1)
list3 = copy.deepcopy(list1)
list3[1][0] = 200
print(f'{list1=}')
print(f'{list3=}')
print(id(list1))
print(id(list3))
```

```
[[1], [2], 3, 4]
list1=[[1], [2], 3, 4]
list3=[[1], [200], 3, 4]
134127526277120
134127352697792
```

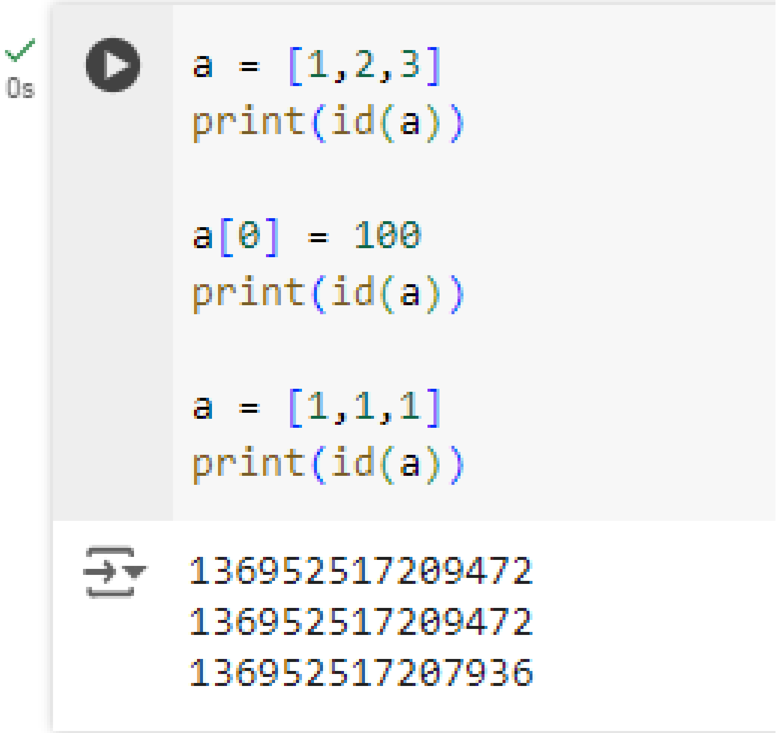
Immutable vs Mutable

- Mutable: The value of the object can be changed, such as lists, dictionaries, and sets.
- Immutable: The value of the object cannot be changed once it is created, such as strings, tuples, and integers.
- The python id() function is used to return a unique identification value of the object stored in the memory. [1]



```
name = 'john'
print(f'{name=}, {id(name)=}')
name = 'jane'
print(f'{name=}, {id(name)=}')
```

```
name='john', id(name)=134128136879536
name='jane', id(name)=134128136879600
```



```
a = [1,2,3]
print(id(a))

a[0] = 100
print(id(a))

a = [1,1,1]
print(id(a))
```

```
136952517209472
136952517209472
136952517207936
```

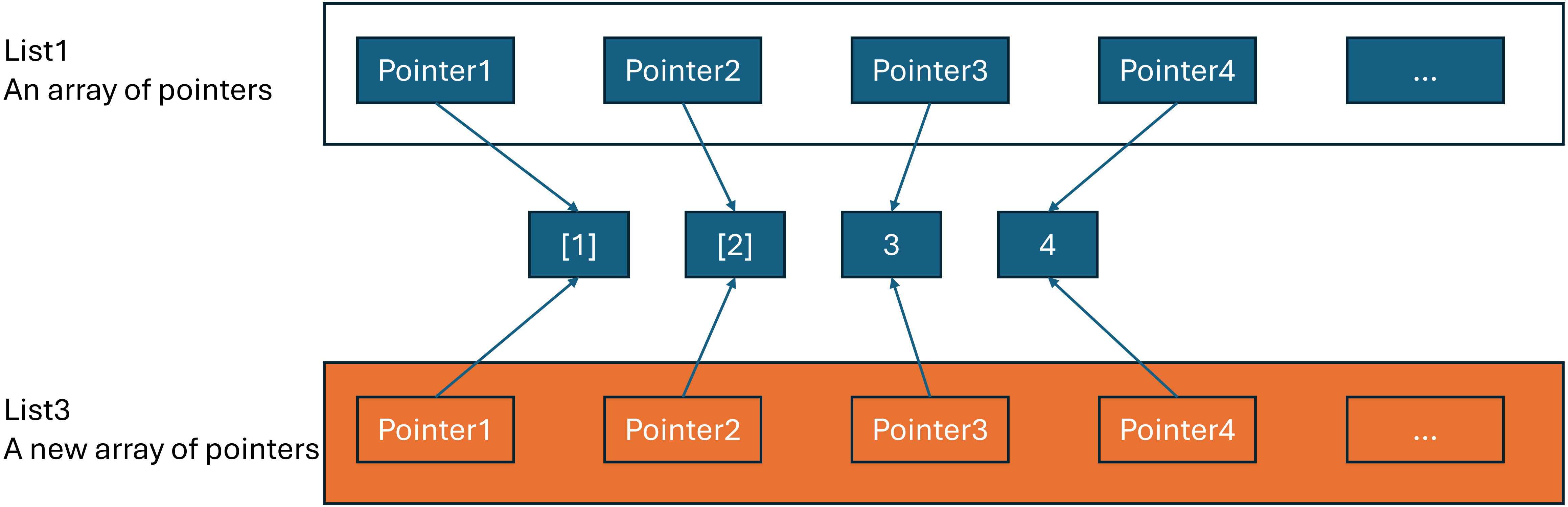
[1] https://www.toppr.com/guides/python-guide/references/methods-and-functions/methods/built-in/python-id/

Shallow Copy

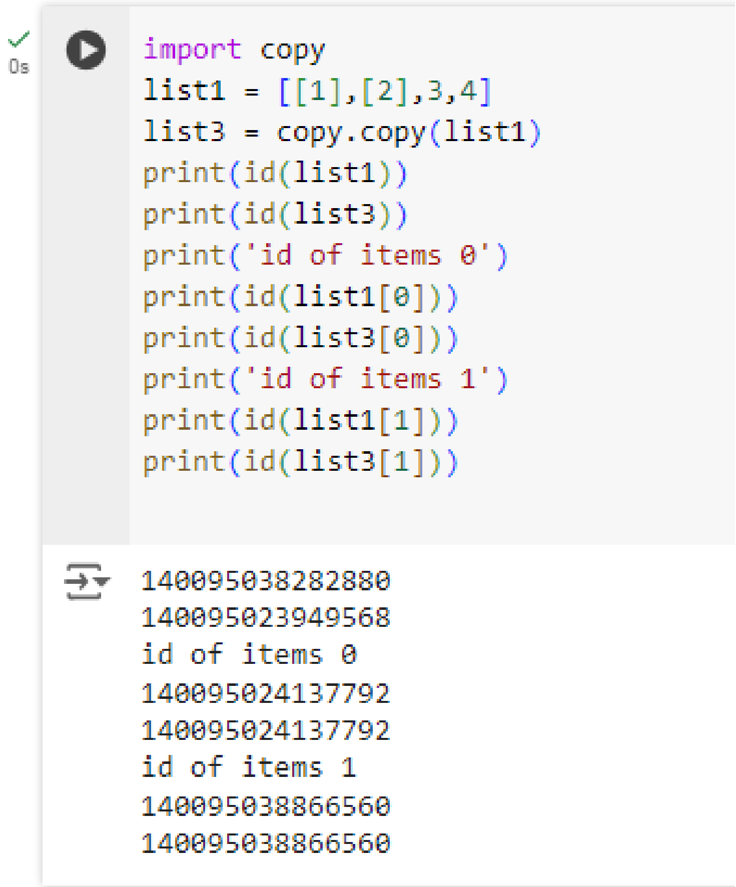
- A shallow copy creates a new object but inserts references into it to the objects found in the original.
- Changes to the copied object can affect the original object if the copied object contains references to mutable objects.
  - A reference is a variable that points to or "references" a location in memory where an object is stored.
  - Zhengchen (list2) and Prof. Zhang (list1)

Shallow Copy

A shallow copy creates a new object but inserts references into it to the objects found in the original.



Shallow Copy



```
import copy
list1 = [[1],[2],3,4]
list3 = copy.copy(list1)
print(id(list1))
print(id(list3))
print('id of items 0')
print(id(list1[0]))
print(id(list3[0]))
print('id of items 1')
print(id(list1[1]))
print(id(list3[1]))
```

```
140095038282880
140095023949568
id of items 0
140095024137792
140095024137792
id of items 1
140095038866560
140095038866560
```

Changes to the copied object can affect the original object if the copied object contains references to mutable objects.

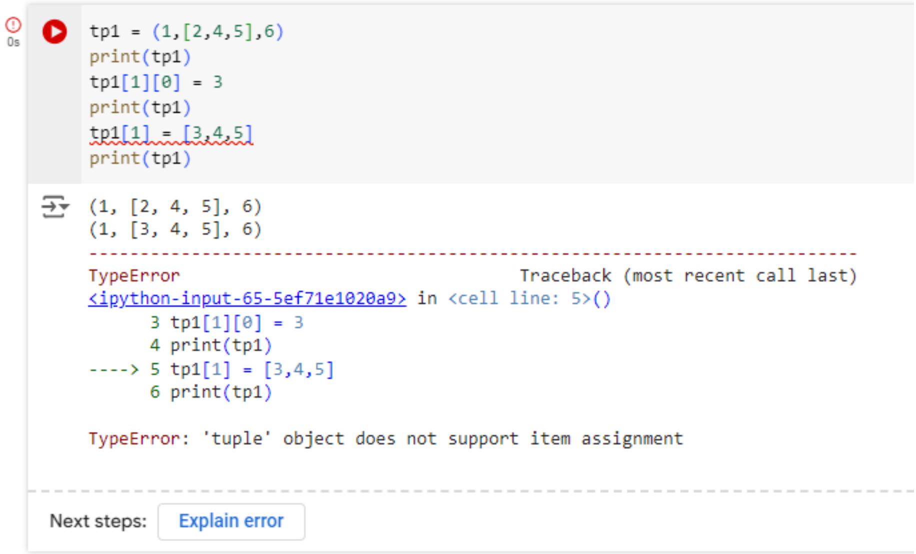
```
list3[0][0] = 100
print('new values')
print(list1)
print(list3)
print('id of the first item')
print(id(list1[0]))
print(id(list3[0]))
Print('\n')
#===============================
list3[0] = [200]
print('new values')
print(list1)
print(list3)
print('id of the first item')
print(id(list1[0]))
print(id(list3[0]))
```

Review

- Operations on a sequence
- Assignment, Deep Copy and Shallow Copy
  - Sometimes you change the values of a list unexpectedly
  - Immutable and Mutable

Tuples

- A sequence of immutable Python objects
- Difference between tuple and list
  - Tuples use parentheses () and lists use square brackets []
  - The tuples can not be changed



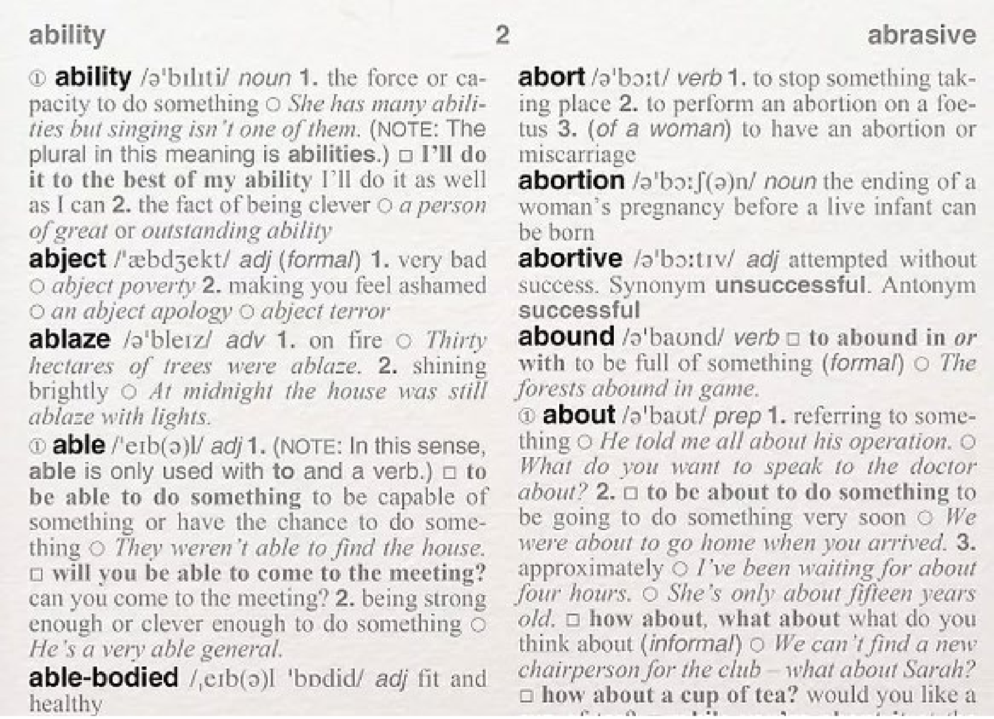
```
tp1 = (1,[2,4,5],6)
print(tp1)
tp1[1][0] = 3
print(tp1)
tp1[1] = [3,4,5]
print(tp1)
```

```
(1, [2, 4, 5], 6)
(1, [3, 4, 5], 6)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-65-5ef71e1020a9> in <cell line: 5>()
      3 tp1[1][0] = 3
      4 print(tp1)
----> 5 tp1[1] = [3,4,5]
      6 print(tp1)

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

Dictionary

- A word and its explanation
- In Python, a dictionary is a built-in data type that stores data in key-value pairs.[2]
- Each key in a dictionary is unique and maps to a value.
  - Example:

ID	Name	Score
001	John	100+
002	John	80
…		



ability 2 abrasive

① **ability** /ə'bɪlɪti/ *noun* 1. the force or capacity to do something ○ *She has many abilities but singing isn't one of them.* (NOTE: The plural in this meaning is **abilities**.) □ **I'll do it to the best of my ability** I'll do it as well as I can 2. the fact of being clever ○ *a person of great* or *outstanding ability*

abject /'æbdʒekt/ *adj* (*formal*) 1. very bad ○ *abject poverty* 2. making you feel ashamed ○ *an abject apology* ○ *abject terror*

ablaze /ə'bleɪz/ *adv* 1. on fire ○ *Thirty hectares of trees were ablaze.* 2. shining brightly ○ *At midnight the house was still ablaze with lights.*

① **able** /'eɪb(ə)l/ *adj* 1. (NOTE: In this sense, **able** is only used with **to** and a verb.) □ **to be able to do something** to be capable of something or have the chance to do something ○ *They weren't able to find the house.* □ **will you be able to come to the meeting?** can you come to the meeting? 2. being strong enough or clever enough to do something ○ *He's a very able general.*

able-bodied /ˌeɪb(ə)l 'bɒdid/ *adj* fit and healthy

abort /ə'bɔːt/ *verb* 1. to stop something taking place 2. to perform an abortion on a foetus 3. (*of a woman*) to have an abortion or miscarriage

abortion /ə'bɔːʃ(ə)n/ *noun* the ending of a woman's pregnancy before a live infant can be born

abortive /ə'bɔːtɪv/ *adj* attempted without success. Synonym **unsuccessful**. Antonym **successful**

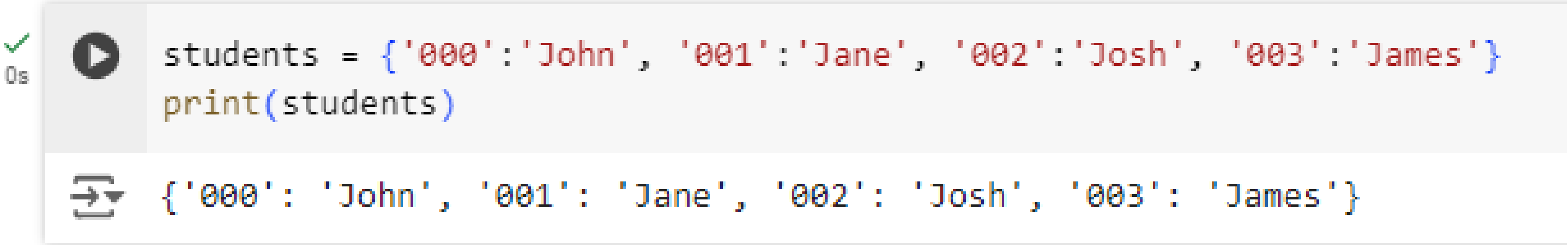
abound /ə'baʊnd/ *verb* □ **to abound in** *or* **with** to be full of something (*formal*) ○ *The forests abound in game.*

① **about** /ə'baʊt/ *prep* 1. referring to something ○ *He told me all about his operation.* ○ *What do you want to speak to the doctor about?* 2. □ **to be about to do something** to be going to do something very soon ○ *We were about to go home when you arrived.* 3. approximately ○ *I've been waiting for about four hours.* ○ *She's only about fifteen years old.* □ **how about, what about** what do you think about (*informal*) ○ *We can't find a new chairperson for the club – what about Sarah?* □ **how about a cup of tea?** would you like a

1. https://www.wikihow.com/Use-a-Dictionary
2. https://tutorialspoint.com/python/python_dictionary.htm

Dictionary

- Each key is separated from its value by a colon (:)
- The items are separated by commas
- The whole thing is enclosed in curly braces
    - An empty dictionary is {}

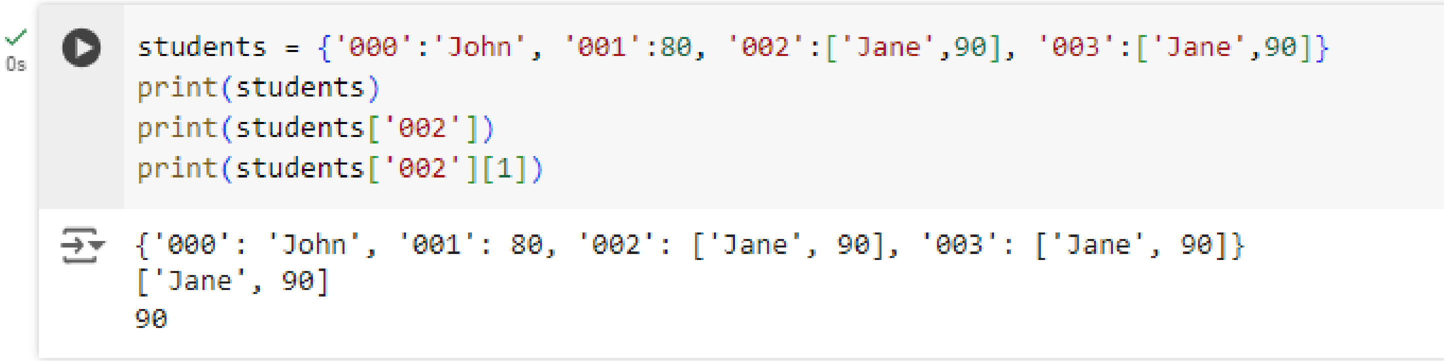


```
students = {'000':'John', '001':'Jane', '002':'Josh', '003':'James'}
print(students)
```

```
{'000': 'John', '001': 'Jane', '002': 'Josh', '003': 'James'}
```

Dictionary

- Keys are unique within a dictionary
- Values may not be unique
- Values can be of any type
  - Strings, numbers, tuples, lists, dictionaries etc.
- Keys must be of an immutable data type
  - Strings, numbers, or tuples

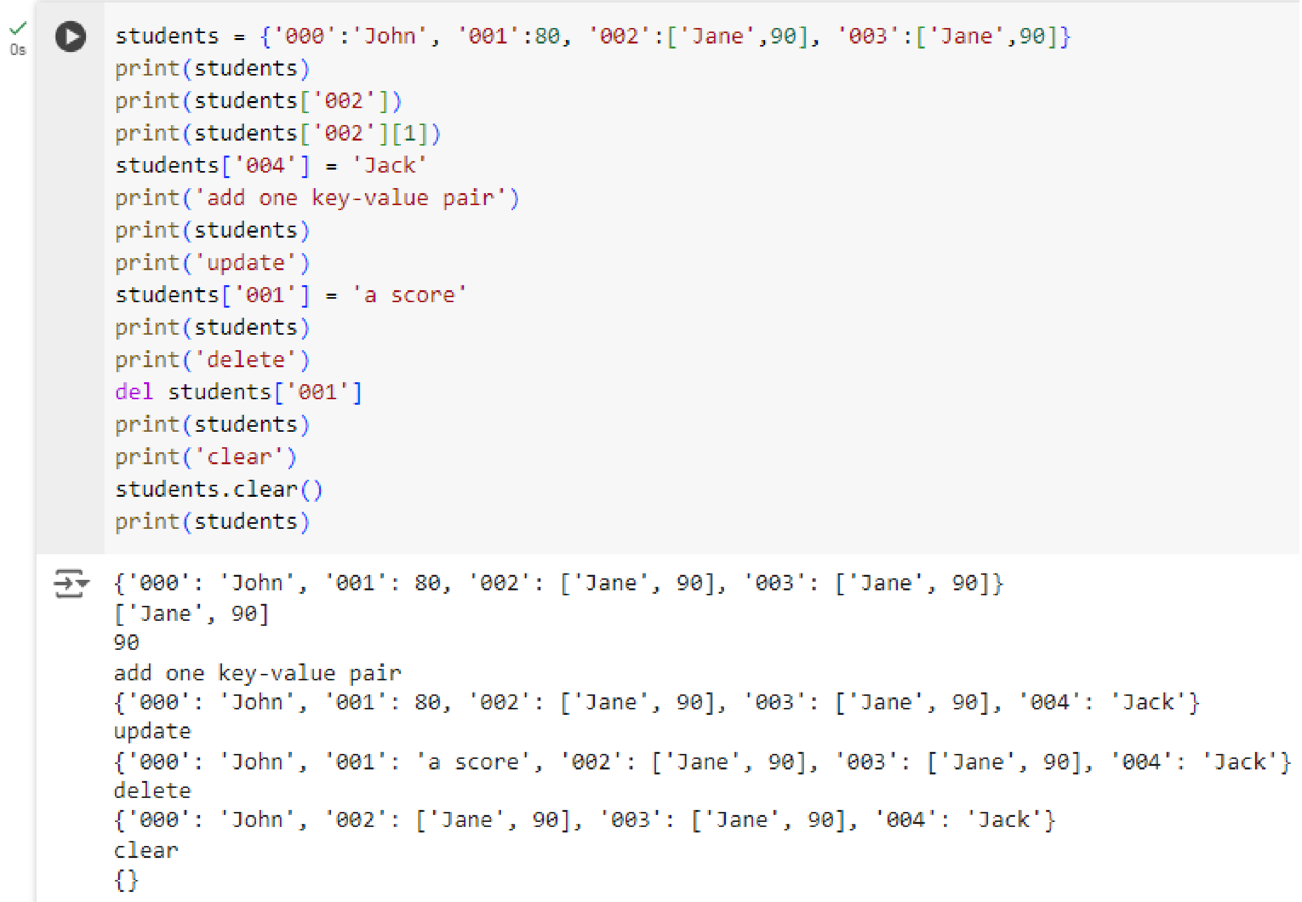


```
students = {'000':'John', '001':80, '002':['Jane',90], '003':['Jane',90]}
print(students)
print(students['002'])
print(students['002'][1])
```

```
{'000': 'John', '001': 80, '002': ['Jane', 90], '003': ['Jane', 90]}
['Jane', 90]
90
```

Dictionary

- Access
  - value = students[key]
- Update and Add
  - students[key] = value
  - If a key already exists, the old value will be overwritten by the new value.
  - If it is a new key, the key and value pair will be added to the dictionary
- Delete
  - del students[key]
  - students.clear()



```
students = {'000':'John', '001':80, '002':['Jane',90], '003':['Jane',90]}
print(students)
print(students['002'])
print(students['002'][1])
students['004'] = 'Jack'
print('add one key-value pair')
print(students)
print('update')
students['001'] = 'a score'
print(students)
print('delete')
del students['001']
print(students)
print('clear')
students.clear()
print(students)
```

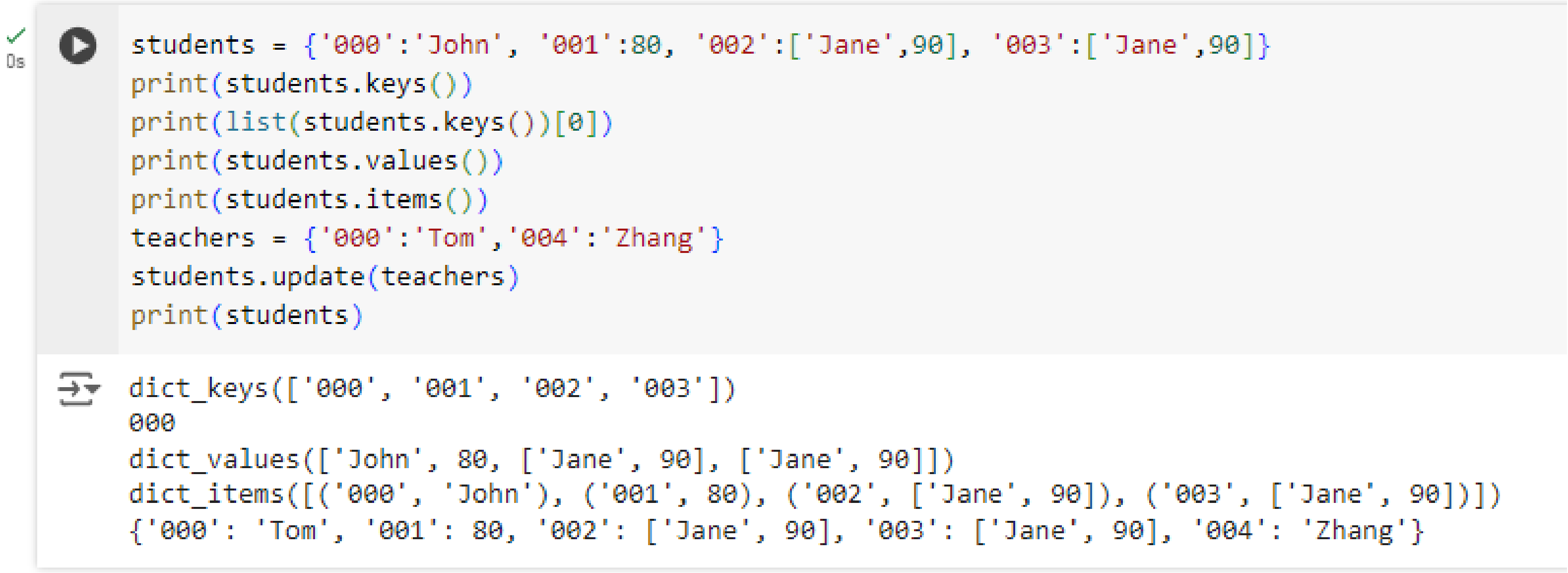
```
{'000': 'John', '001': 80, '002': ['Jane', 90], '003': ['Jane', 90]}
['Jane', 90]
90
add one key-value pair
{'000': 'John', '001': 80, '002': ['Jane', 90], '003': ['Jane', 90], '004': 'Jack'}
update
{'000': 'John', '001': 'a score', '002': ['Jane', 90], '003': ['Jane', 90], '004': 'Jack'}
delete
{'000': 'John', '002': ['Jane', 90], '003': ['Jane', 90], '004': 'Jack'}
clear
{}
```

Built-in Functions with Dictionaries

Name	Description
len(dict)	Give the length of the dictionary, which is the number of items in the dictionary
str(dict)	Produce a printable string representation of a dictionary

Python Dictionary Methods

Name	Description
dictionary.keys()	Returns list of dictionary keys
dictionary.values()	Returns list of dictionary values
dictionary.items()	Returns list of dictionary items: (key, value) tuple pairs
dictionary.update(dictionary2)	Adds dictionary2's key-values pairs to the first dictionary



```python
students = {'000':'John', '001':80, '002':['Jane',90], '003':['Jane',90]}
print(students.keys())
print(list(students.keys())[0])
print(students.values())
print(students.items())
teachers = {'000':'Tom','004':'Zhang'}
students.update(teachers)
print(students)
```

```
dict_keys(['000', '001', '002', '003'])
000
dict_values(['John', 80, ['Jane', 90], ['Jane', 90]])
dict_items([('000', 'John'), ('001', 80), ('002', ['Jane', 90]), ('003', ['Jane', 90])])
{'000': 'Tom', '001': 80, '002': ['Jane', 90], '003': ['Jane', 90], '004': 'Zhang'}
```

For loop

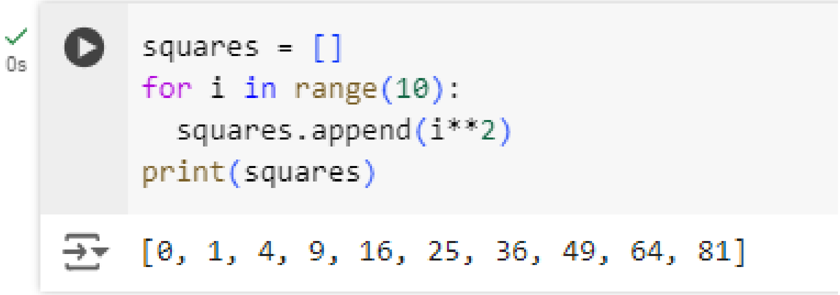
- Practice
  - Create a dictionary, which contains student scores
  - Key is the student number
  - Value is the score
  - Print the sorted scores

Review

- Key-value pairs
- Keys are unique
- Add, update, delete
- for key, value in my_dict.items():
- keys(), values()

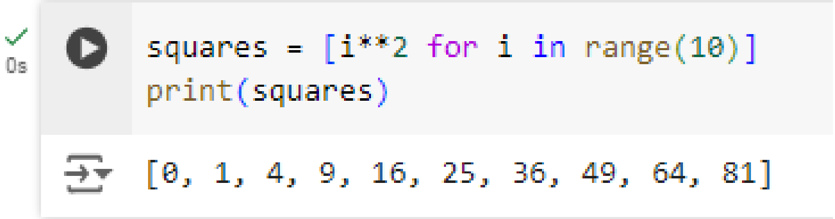
List Comprehension

- **Generate a new list** by applying an expression to each item in an **existing iterable** (like a list or range) in **a single line of code**.
    - Create a list containing the first ten perfect squares



```
squares = []
for i in range(10):
  squares.append(i**2)
print(squares)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

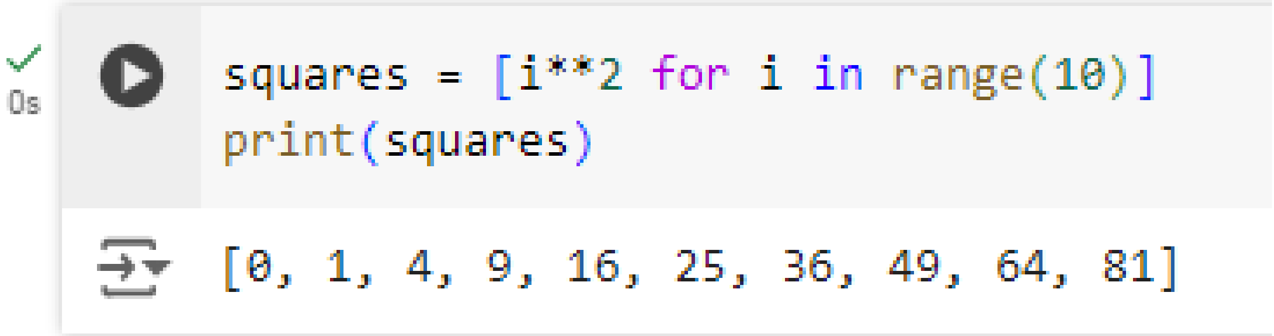


```
squares = [i**2 for i in range(10)]
print(squares)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

List Comprehension

- new_list = [expression for member in iterable]
  - expression: the list member itself or any valid expression returns a value
    - i**2
  - member: the object or value in the list or iterable
    - i
  - iterable: a list, set, sequence, generator or any other object that can return its elements one at a time
    - list(range(10)) : [0,1,2,3,4,5,6,7,8,9]

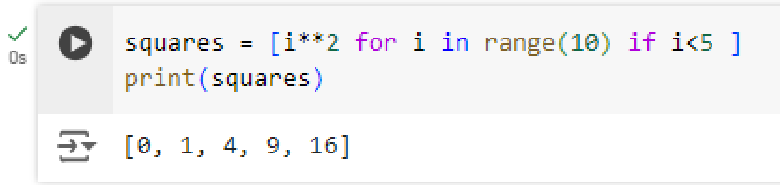


```
squares = [i**2 for i in range(10)]
print(squares)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

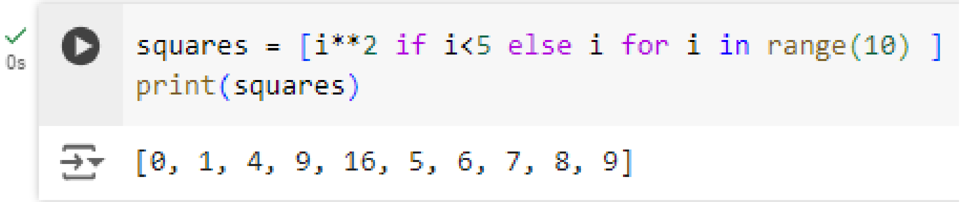
List Comprehension

- Using conditional logic
- new_list= [expression for member in iterable (if conditional)]



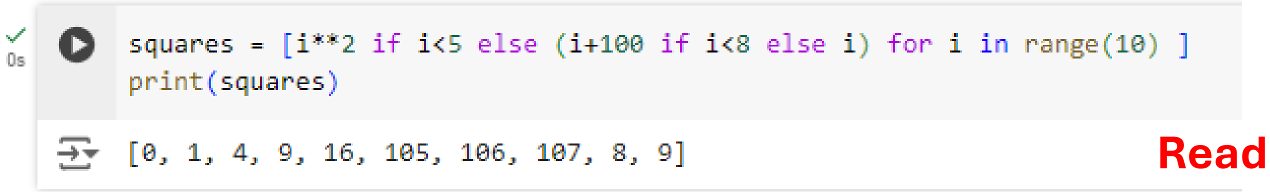
```
squares = [i**2 for i in range(10) if i<5 ]
print(squares)
```

```
[0, 1, 4, 9, 16]
```



```
squares = [i**2 if i<5 else i for i in range(10) ]
print(squares)
```

```
[0, 1, 4, 9, 16, 5, 6, 7, 8, 9]
```

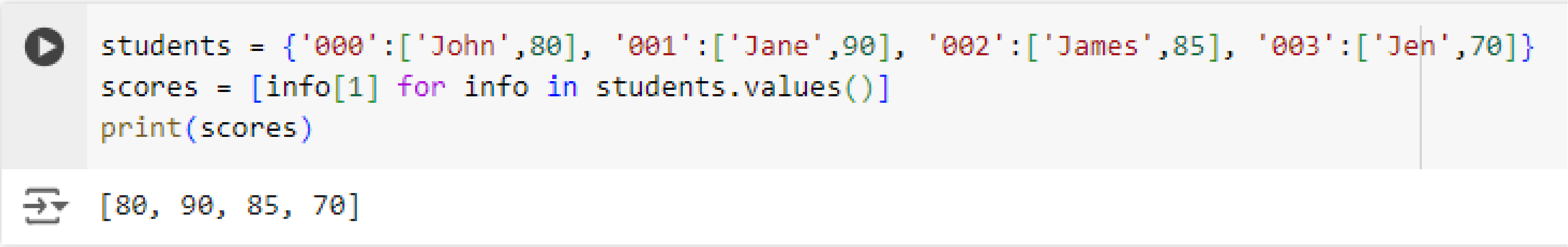


```
squares = [i**2 if i<5 else (i+100 if i<8 else i) for i in range(10) ]
print(squares)
```

```
[0, 1, 4, 9, 16, 105, 106, 107, 8, 9]
```

Readability

Integrated application



```
students = {'000':['John',80], '001':['Jane',90], '002':['James',85], '003':['Jen',70]}
scores = [info[1] for info in students.values()]
print(scores)
```

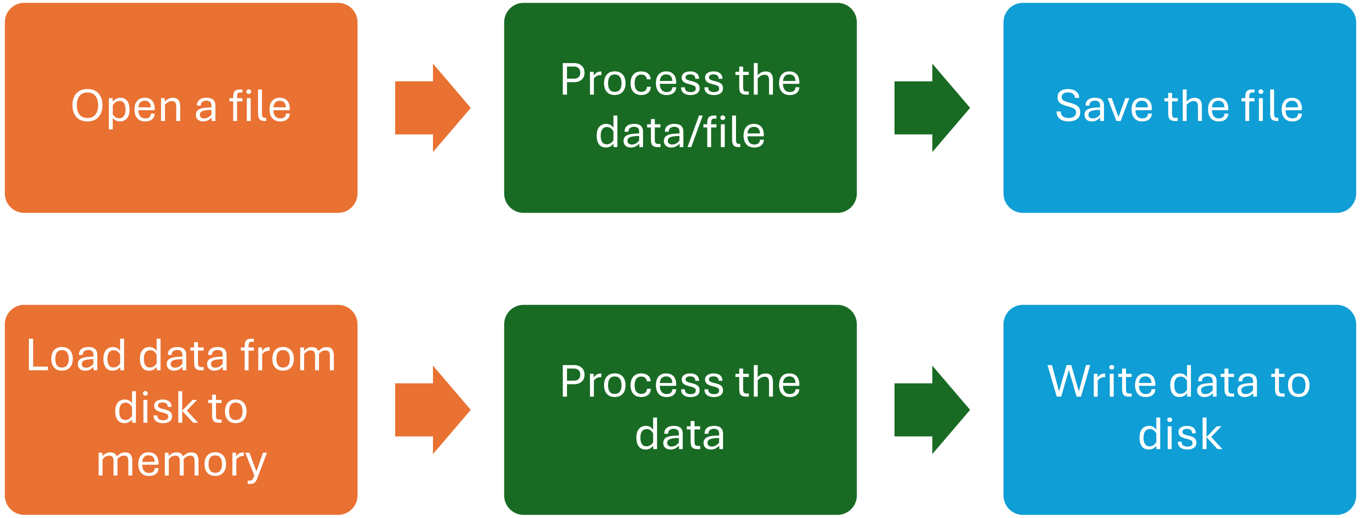
```
[80, 90, 85, 70]
```

Review

- new_list= [expression for member in iterable (if conditional)]
- Efficiency sometimes comes at the cost of readability

Files I/O

- Why do we need a file?
  - Data used in the program is stored in the memory
  - Still can find the data after reboot the computer/restart the program
  - Keep the data permanent
- File operations



Open a file → Process the data/file → Save the file

Load data from disk to memory → Process the data → Write data to disk

Open a file

- Python’s built-in open() function
- Create one file object that can be utilized
- Syntax:
  - file_object= open(file_name[, access_mode] [, encoding])

Object Oriented Programming

- Class
  - A blueprint for creating objects
  - Defines a set of attributes and methods
  - Groups data and behavior together in a reusable and organized way
- Object
  - An instance of a class
  - Key Characteristics
    - Attributes (or Properties): Variables that store data specific to the object
      - Name, Age
    - Methods (or Functions): Functions that define the behavior of the object
      - Walk, Speak

open()

- File_name
  - a string value that contains the name of the file that you want to access
- Access_mode
  - the mode in which the file to be opened, i.e., read, write, append, etc.
  - Optional, the default model is read(r)
- Encoding
  - Default value depends on your operating system
  - utf-8
  - Remember the Unicode thing?

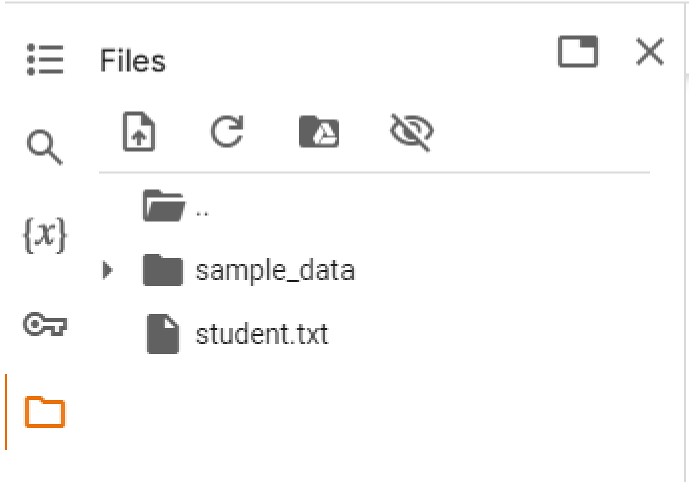
mode

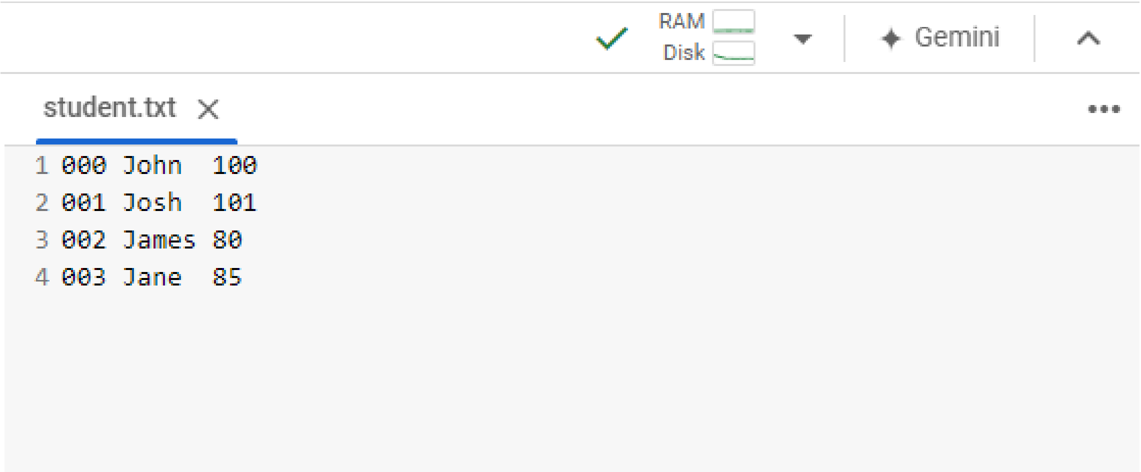
- Open('student_info.txt', mode='r')

Mode	Description
r	Opens a file for reading only. This is the default mode.
r+	Opens a file for both reading and writing.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	Opens a file for appending. If the file does not exist, it creates a new file for writing.
b	rb, wb, ab, etc. Opens the file in binary mode.
…	

Practice

- Create a new file on Google Colab
- Double click 'student.txt' and edit it, and save it (Ctrl + S)

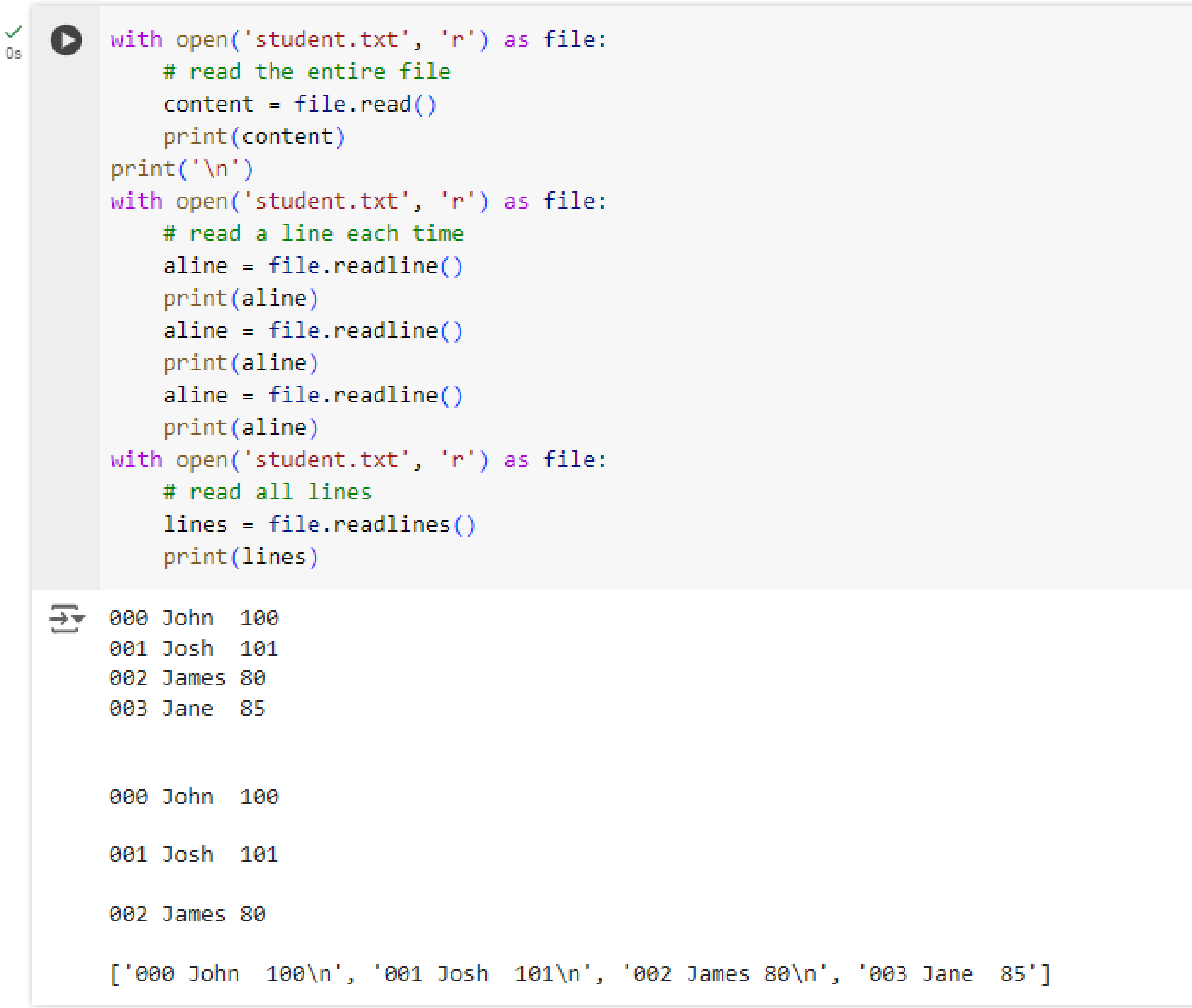




```
000 John  100
001 Josh  101
002 James 80
003 Jane  85
```

File pointer

- Indicate the current focused position of the file
- Description of mode 'r'
  - Opens a file for reading only.
  - The file pointer is placed at the beginning of the file.

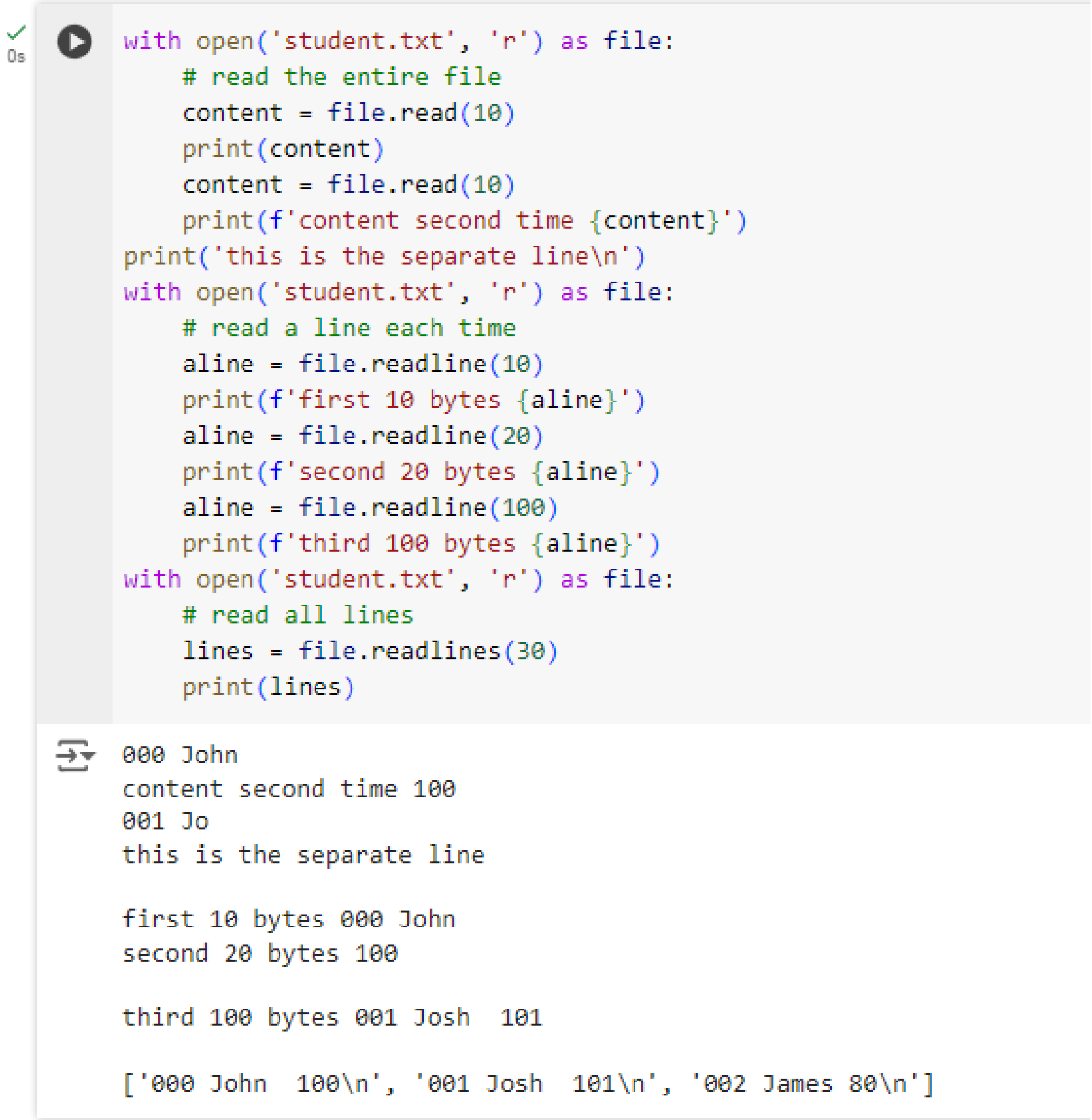


```
with open('student.txt', 'r') as file:
    # read the entire file
    content = file.read()
    print(content)
print('\n')
with open('student.txt', 'r') as file:
    # read a line each time
    aline = file.readline()
    print(aline)
    aline = file.readline()
    print(aline)
    aline = file.readline()
    print(aline)
with open('student.txt', 'r') as file:
    # read all lines
    lines = file.readlines()
    print(lines)
```

```
000 John  100
001 Josh  101
002 James 80
003 Jane  85


000 John  100

001 Josh  101

002 James 80

['000 John  100\n', '001 Josh  101\n', '002 James 80\n', '003 Jane  85']
```

Read a file

- read(size)
  - returns the specified number of bytes from the file.
  - size
    - Indicates the number of bytes to read from the file.
    - Optional, default is -1 which means the whole file
    - If omitted or set to a negative value, the method reads until the end of the file.
- readline(size)
  - returns one line from the file
  - If size < len(aline)
  - If size > len(aline)
- readlines(hint)
  - returns a list containing each line in the file as a list item.
  - If returned number of bytes > hint, then stop

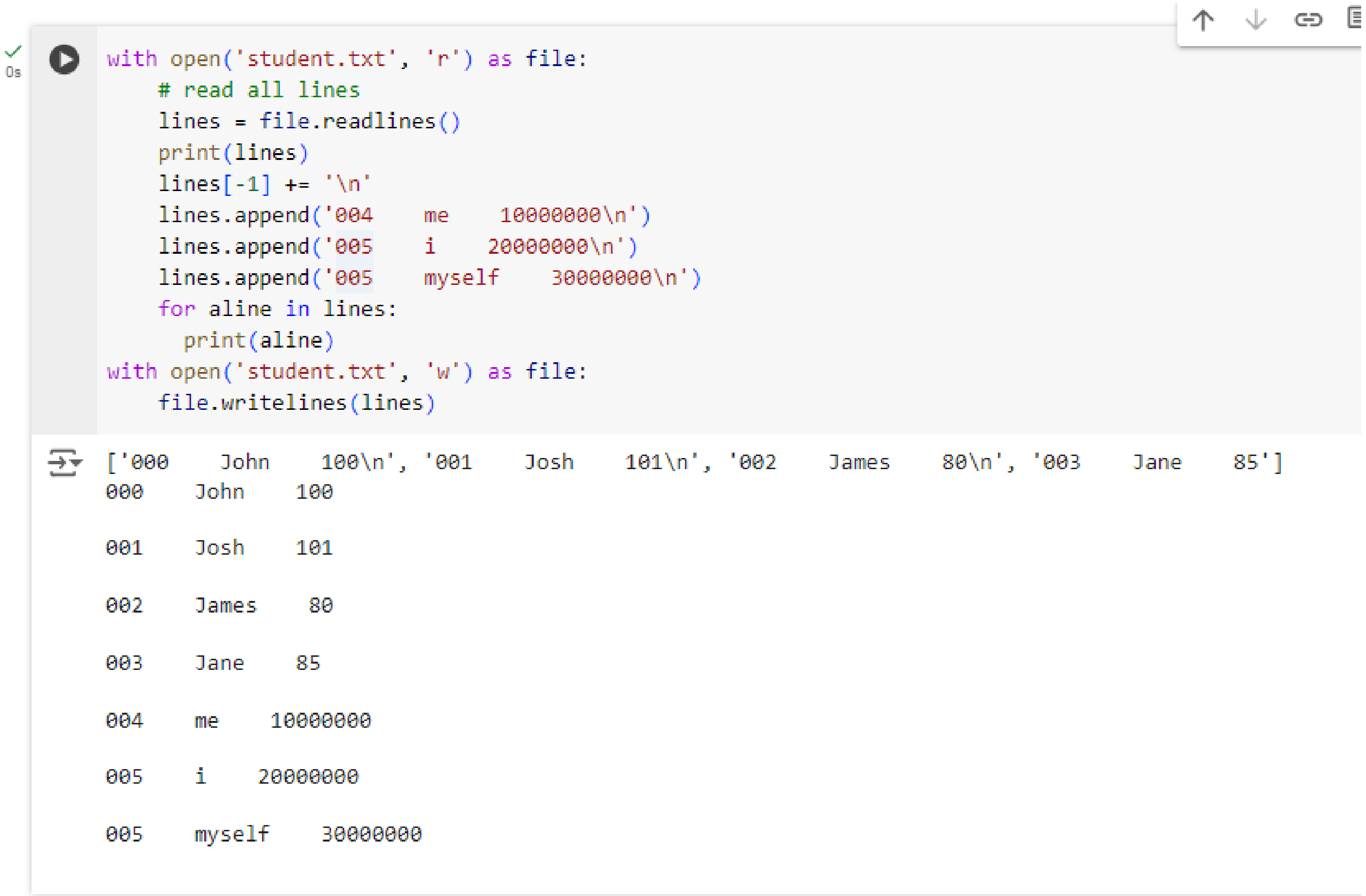


```
with open('student.txt', 'r') as file:
    # read the entire file
    content = file.read(10)
    print(content)
    content = file.read(10)
    print(f'content second time {content}')
print('this is the separate line\n')
with open('student.txt', 'r') as file:
    # read a line each time
    aline = file.readline(10)
    print(f'first 10 bytes {aline}')
    aline = file.readline(20)
    print(f'second 20 bytes {aline}')
    aline = file.readline(100)
    print(f'third 100 bytes {aline}')
with open('student.txt', 'r') as file:
    # read all lines
    lines = file.readlines(30)
    print(lines)
```

```
000 John
content second time 100
001 Jo
this is the separate line

first 10 bytes 000 John
second 20 bytes 100

third 100 bytes 001 Josh  101

['000 John  100\n', '001 Josh  101\n', '002 James 80\n']
```

Update data

- Take note of the \n at the end of the line
- Try to make the lines aligned using f-string

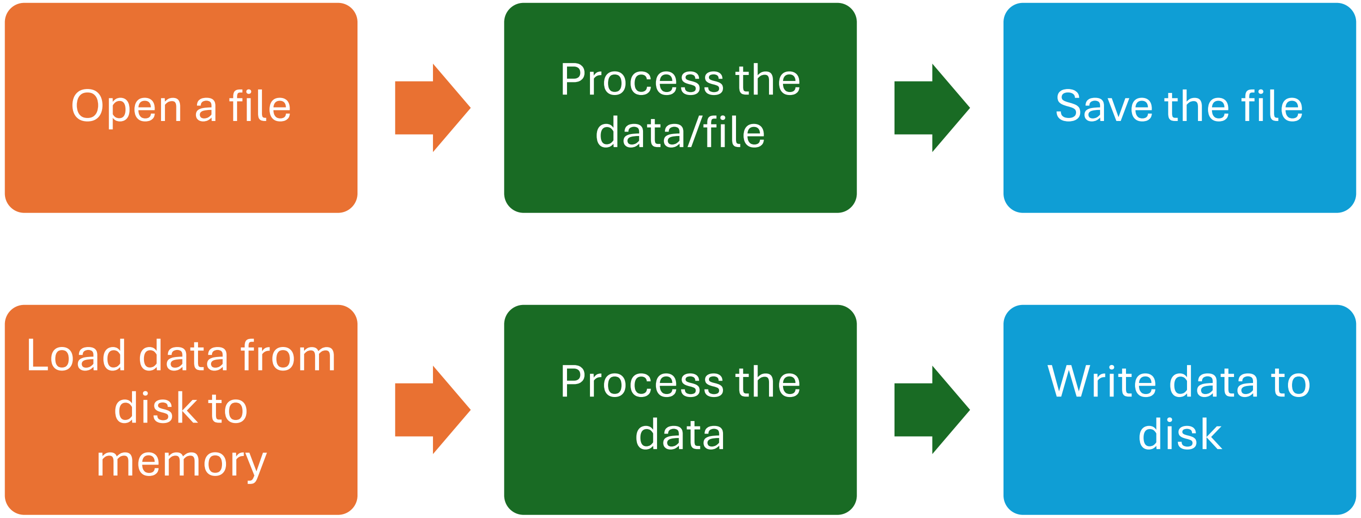


```
with open('student.txt', 'r') as file:
    # read all lines
    lines = file.readlines()
    print(lines)
    lines[-1] += '\n'
    lines.append('004    me    10000000\n')
    lines.append('005    i    20000000\n')
    lines.append('005    myself    30000000\n')
    for aline in lines:
      print(aline)
with open('student.txt', 'w') as file:
    file.writelines(lines)
```

```
['000    John    100\n', '001    Josh    101\n', '002    James    80\n', '003    Jane    85']
000    John    100

001    Josh    101

002    James    80

003    Jane    85

004    me    10000000

005    i    20000000

005    myself    30000000
```

Write data to a file

- write(str)
  - This is the String to be written in the file.
- writelines(sequence)
  - This is the Sequence of the strings.



With open() as file:

- The with statement is a context manager
- Automatically handles file opening and closing
- Ensures the file is properly closed even if an exception occurs
- Restricts the variable scope to the with block

The 'with' statement

- Advantages of the with statement:
    - More concise
    - Automatic resource management, avoiding resource leaks
    - Safer, more readable code
    - Restricts the variable scope, reducing potential bugs
- Disadvantages of the traditional method:
    - Requires explicit file closing
    - Error-prone, can lead to resource leaks
    - Variable remains in scope, which may cause unexpected issues

Review

- Read data from a file
- Operations on the data
- Write to a file
- File Pointer
- 'with' statement

Topics to Carry Over to Lectures 4 and 5

- Return value of a function
    - Returns one line of the file
- Bytes Data Type
    - ASCII, Byte, bit, Unicode
    - Beyond txt file: image, audio, video, npy

Review

- String, List
  - sequence
- **Immutable and mutable**
  - Tuple
  - Shallow Copy and Deep Copy
- Dictionary
- File I/O