3. Containers:

3.1 Lists and their methods
3.2 Tuples and their methods
3.3 Sets and their methods
3.4 Dictionaries and their methods

3.1 Lists

گاهی اوقات نیاز می‌شود تا چندین داده را در یک متغیر ذخیره کرد برای این کار از لیست ها استفاده میشود
برای ایجاد یک لیست کافیست تا داده ها را توسط یک , از هم جدا کرده و در بین [ ] قرار دهید:

```
My_list=[12,23,21,45,58,10]
```

لیست ها میتوانند از هر نوع داده برای ذخیره سازی پشتیبانی کنند

```
My_list_2=[True,"Python",12,-1.5,"char"]
```

لیست ها منظم اند و ترتیبشان حفظ میشود و اعضای یک لیست هر کدام یک شاخص (عدد) دارند که به آن (index) میگویند و نمایانگر مکان آن عضو در لیست است
در پایتون این شاخص ها از 0 شروع میشود پس در مثال My_list اعداد 12 23 به ترتیب دارای ایندکس 0 و 1 هستند به همین منوال 21 ایندکس 2 و 45 ایندکس 3 و ... دارند

برای دسترسی به اعضای لیست ها کافیست ایندکس مورد نظر را بدانیم به این صورت که :

My_list_2[3] → -1.5 My_list_2[1] → "Python"

منظور از نظم لیست ها این است که مکان اعضا عوض نمیشوند و اگر عضوی به اعضا اضافه شود به انتهای لیست افزوده خواهد شد البته روش هایی برای عوض کردن این ترتیب وجود.

Index لیست ها از انتها نیز قابل شمارش اند بدین صورت که :

My_list=[12,23,21,45,58,10]

Index	Data	Index	Data
0	12	-6	12
1	23	-5	23
2	21	-4	21
3	45	-3	45
4	58	-2	58
5	10	-1	10

برش لیست ها (list slicing)

گاهی نیاز است به جای یک عضو با چندین عضو سر و کار داشته باشیم
برای دسترسی به عضو های دلخواه از برش لیست ها استفاده میکنیم و قسمت های به اصطلاح بریده شده در اختیار ما قرار میگیرند
توجه شود که در برش لیست به خود لیست آسیبی نمیرسد

```
My_list=[12,23,21,45,58,10,-1,-2]
My_list [      x          :          Y          :          z          ]
```

نکته:

- Y میتواند خالی بماند در آن صورت برش تا انتهای لیست خواهد بود
- همواره انتهای برش y-1 است
- z به طور پیشفرض 1 است اما میتواند هر عدد صحیحی باشد مثل 2 و 3 و 1- و 2-
- در صورت منفی بودن z برش به صورت معکوس صورت خواهد پذیرفت پس طبیعتا تغییرات لازم برای xوy باید اعمال شود

- My_list[len(My_list) : 0 : -1]

- تابع len() طول یا تعداد اعضای یک متغیر را نشان می‌دهد

چند نکته

لیست ها تغییر پذیر اند به این معنا که پس از تشکیل یک لیست در برنامه قابلیت تغییر کم و زیاد کردن اعضای آن وجود دارد که در آینده درباره آن بحث خواهد شد

از آنجا که داده ها در لیست ها شاخص دهی می‌شوند پس وجود داده های تکراری در لیست مشکلی به وجود نخواهد آورد

اگر تعدادی متغیر داشته باشیم با دستور list(()) میتوان آنها را به لیست تبدیل کرد درست مانند توابع int() و float() و str()
متغیر ها باید همگی داخل یک پرانتز یا کروشه یا براکت به این تابع داده شوند

```
My_list = list((x1 , x2 , x3, ...))
```

مثال:
برنامه ای بنویسید که سه عدد از کاربر دریافت کرده و میانگین آنها را محاسبه کند (با استفاده از ایندکس دهی و تابع list())

تغییر داده های یک لیست

همانطور که گفته شد داده های یک لیست می‌توانند تغییر کنند برای انجام این کار:
ایندکس مورد نیاز را فراخوانی کرده و مقدار جدید را به آن منصوب میکنیم

```
My_list= ["Benz", "BMW", "Audi", "cherry"]
My_list [1] = "Ford"
```

همچنین می‌توان تعدادی از اعضا را تغییر داد با استفاده از برش لیست

```
My_list [1:3] = ["Ford", “Rolls Royce"]
```

نکته : لزومی به برابر بودن تعداد اعضای برش شده با اعضای جایگزین کننده نیست

Lists methods

تا اینجای کار با برخی از توابع مثل print و input و len آشنا شدیم که در هر جایی از برنامه می‌توانیم از آنها استفاده کنیم

اما نوع دیگری از توابع وجود دارند که مخصوص متغیر های خاصی هستند و برای هر متغیری نمیتوان استفاده کرد که به این توابع متد (method) گفته می‌شود
برای مثال str ها و list ها و ... هر یک متد های خاص خود را دارند

تفاوت دیگر توابع و متد ها در نحوه فراخوانی آنهاست :
در استفاده از توابع نام تابع را فراخوانی کرده و مقداری را به عنوان ورودی داخل پرانتز قرار میدهیم

int(23e6) print(“Hello world”) len(My_list)

اما در استفاده از متد ها از . استفاده کرده و آنها را روی داده خاصی صدا میزنیم مثل:

My_list.sort() My_list.reverse() My_list.append()

برخی متد ها ورودی قبول کرده و برخی دیگر نیز نیازی به ورودی گرفتن ندارند

برخی متد های پر کاربرد لیست ها

متد	توضیح	سینتکس
append()	عنصر ورودی اش را به انتهای لیست اضافه میکند	My_list.append(element)
count()	تعداد تکرار یک عنصر مشخص را در یک لیست بر میگرداند	My_list.count(element)
extend()	عناصر یک لیست خاص را به انتهای یک لیست دیگر اضافه میکند	My_list.extend(My_list_2)
index()	ایندکس ورودی اش را در یک لیست پیدا کرده و بر میگرداند در صورت تکرار عنصر ورودی در لیست کمترین ایندکس موجود را بر میگرداند	My_list.index(element)
insert()	عنصر y را در ایندکس x قرار می‌دهد	My_list.insert(x,y)
pop()	ایندکس ورودی را از لیست حذف میکند	My_list.pop(position)
sort()	عناصر لیست را بر اساس حروف الفبا یا ترتیب عددی مرتب میکند	My_list.sort()

نکته: متد sort() پارامتری اختیاری نیز دریافت میکند که در صورت استفاده، لیست به صورت معکوس مرتب میشود
My_list.sort(reverse=True)

به کوچک بودن نام متد ها دقت کنید

منابع بیشتر

برای بررسی متد های بیشتر به مستندات پایتون در سایت رسمی پایتون مراجعه کنید

https://docs.python.org/3/tutorial/datastructures.html#more-on-lists

3.2 Tuples

همانند لیست ها تاپل ها نیز برای ذخیره تعداد زیادی از داده ها در یک متغیر استفاده میشود

تاپل ها نیز همانند لیست ها منظم هستند و ایندکس پذیر اند
اما تفاوت تاپل ها با لیست ها در تغییر ناپذیری آنهاست (immutable) به این معنا که پس از ساخت قابلیت تغییر دادن آن وجود نخواهد داشت

نحوه ساخت تاپل : با استفاده از ()

تک عضوی My_tuple=(element,)

چند عضوی My_tuple=(elements)

در ساخت تاپل تک عضوی در صورت نذاشتن , متغیر شما تاپل نخواهد بود بلکه بسته به مقدار آن نوع مختلف خواهد داشت (str,int,float,...)

تاپل ها همانند لیست ها با فراخوانی ایندکس هایشان قابل دسترسی هستند

همچنین مانند لیست ها میتوان داده هایی از انواع مختلف را در یک تاپل ذخیره کرد

```
Mytuple=(12,True,0.0,"char")
```

باز هم مانند لیست ها با استفاده از تابع tuple() میتوان جمعی از متغیر ها را به یک تاپل تبدیل کرد

برای مثال اگر فرض کنیم سه متغیر x و y و z داریم :

```
my_tuple = tuple((x,y,z))
```

به پرانتز های دوگانه دقت کنید
برش دهی در تاپل ها نیز درست همانند لیست ها عمل میکند

تغییر تاپل ها

علیرغم تغییر ناپذیری تاپل ها با استفاده از روش هایی میتوان اعضای تاپل ها را در صورت نیاز تغییر داد راه حل استفاده از لیست ها است
بدین ترتیب که یک کپی از تاپل در قالب لیست تهیه کرده و تغییرات لازم را در لیست به وجود آورده و در نهایت لیست تغییر یافته را با دستور tuple() دوباره به متغیر مورد نظر منصوب میکنیم

```
Mytuple = (“BMW” , “Benz” , “Ford”)

Mylist = list (Mytuple)

Mylist = Mylist.append(“Rolls Royce”)    یا  Mylist[2] = “Rolls Royce”

Mytuple = tuple(Mylist)
```

بدین ترتیب میتوان با این روش از متد های لیست ها نیز بر تاپل ها استفاده کرد

راه دیگر اضافه کردن داده ای به تاپل ها استفاده از عملگر + است Tuple1 + tuple2

از عملگر * نیز در تاپل ها استفاده میشود در این صورت داده های داخل تاپل به همان ترتیبی که هستند چند برابر میشود Mytuple * 3

برای پاک کردن کامل یک تاپل و از بین بردن آن از del استفاده میکنند(بدون پرانتز) del tuple1

تاپل ها فقط دارای دو متد هستند که با آنها در لیست ها آشنا شدیم

count()	تعداد تکرار یک عنصر مشخص را در یک تاپل بر میگرداند	My_tuple.count(element)
index()	ایندکس ورودی اش را در یک تاپل پیدا کرده و بر میگرداند در صورت تکرار عنصر ورودی در تاپل کمترین ایندکس موجود را بر میگرداند	My_tuple.index(element)

Packing و unpacking یک تاپل

پک کردن یک تاپل به این معناست که به یک متغیر تعدادی داده به صورت یک تاپل مقدار دهیم
در واقع به زبان دیگر همان تشکیل یک تاپل
آنپک کردن معنای معکوس این عمل را می‌دهد یعنی شکستن یک تاپل و انتصاب مقادیر موجود به متغیر های دیگر

فرض کنید یک تاپل با محتویات زیر داریم :

Mytuple =(”River” , ”Lego” , “Taxi”, “Tree” , ”Cello”)

میخواهیم داده ها را بر حسب جنس ماده در متغیر های جدا unpack کنیم

(Water , Plastic , Metal , Wood , Wood2)=Mytuple

بدین ترتیب با پرینت کردن هر یک از این متغیر ها عنصر متناظر با تاپل Mytuple چاپ خواهد شد
اما همچنان Mytuple وجود خواهد داشت و آسیبی به آن نخواهد رسید

میتوان از * استفاده کرد تا چند داده را به صورت یک لیست به یک متغیر unpack کرد مثل wood و wood 2

(Water , Plastic , Metal , *Wood)=Mytuple

فرض کنیم :

Mytuple =(”River” , “Tree” , ”Cello”, ”Lego”)

در این صورت

(Water , *Wood , Plastic)=Mytuple

ایندکس های اول و آخر Mytuple به ترتیب به متغیر های Water و Plastic منصوب می‌شوند
و تمام ایندکس هاس اضافی بین آنها به Wood خواهد رسید

3.3 Sets

ست ها سومین container از چهارنوعی است که مورد برسی ماست
ست ها کاملا بر خلاف لیست ها نا منظم و غیر قابل ایندکس دهی میباشند

نا منظم بودن ست ها به این معناست که در پرینت کردن، اعضای آن غیر قابل پیشبینی و رندوم چاپ خواهند شد

به علت نا منظم بودن ست ها بنا بر این نمیتوان به داده های داخل آن ایندکس داد بنا بر همین دلیل
مقادر تکراری در ست ها قابل قبول نخواهند بود

مانند لیست ها (list((x,y,...)) و تاپل ها (tuple((x,y,...)) ، ست ها نیز با استفاده از تابع (set((x,y,...)) قابلیت packing دارند
و همچنین قابلیت ذخیره سازی انواع مختلف داده ها را همزمان در خود دارند

ست ها با استفاده از {} تولید میشوند

Myset={12 , “true” , False , -3.5}

متد های ست ها

Add()	تنها یک مقدار خاص را به ست اضافه میکند	Myset.add(element)
update()	عناصر یک ست یا لیست یا دیکشنری و ...را به ست دیگر اضافه میکند	Myset.update(iterable_object)
remove()	عنصر داده شنده به این متد از داخل ست حذف خواهد شد	Myset.remove(element)

برای اطلاعات بیشتر به سایت رسمی پایتون مراجعه کنید

8.7. sets — Unordered collections of unique elements — Python 2.7.18 documentation

3.4 Dictionaries

دیکشنری ها برای نگه داشتن داده ها در قالب کلید:مقدار (key:value) استفاده می‌شوند

دیکشنری ها در پایتون همانند دیکشنری های واقعی برای هر مقدار(value) یک کلید (key)،
که حتما از نوع str است در نظر میگیردکه با فراخوانی آن کلید، مقدار متناظر قابل دسترسی خواهد بود

دیکشنری ها با {} تولید میشوند

```
My_dict = {“key1”:value1 , “key2”:value2 , ….}
```

Value ها میتوانند هر نوع داده ای باشند

Value ها میتوانند تکراری باشند اما key ها قابل تکرار نیستند

مقادیر کلید ها به این طریق قابل دسترسی هستند :

```
My_dict[“key”]
```

همچنین دیکشنری ها قابل تغییرند

```
My_dict[“key”] = value
```

اگر کلید وجود داشته باشد مقدار آن با این دستور عوض خواهد شد و اگر وجود نداشته باشد با این دستور چنین کلید و مقداری به دیکشنری اضافه خواهد شد

برای پک کردن تعدادی متغیر در یک دیکشنری از تابع dict() استفاده میکنیم

```
My_dict = dict (key=value , key2=value2 , ….)
```

متد های دیکشنری

keys()	تمامی کلید های موجود در دیکشنری را در یک لیست بر میگرداند	My_dict.keys()
values()	تمامی مقادیر موجود در دیکشنری را در یک لیست بر میگرداند	My_dict.values()
pop()	کلید و مقدار مورد نظر را از دیکشنری حذف میکند	My_dict.pop(“key”)
update()	کلید و مقدار جدید به دیکشنری اضافه میکند	My_dict.update(“key”:value)

برای اطلاعات و آشنایی با متد های دیگر به سایت رسمی پایتون مراجعه کنید

Python Dictionary Methods (w3schools.com)

جمع بندی کلی

Container	Order	Data type	Duplicate	symbol	packing
List	True	Any	Allowed	[ ]	list((x,y,…))
Tuple	True	Any	Allowed	( )	tuple((x,y,…))
Set	False	Any	Not allowed	{ }	set((x,y,…))
Dictionary	True	Any	Allowed(values) not allowed (keys)	{ : }	dict(x=y,…)

عملگر های هویت و عضویت

ادامه مبحث عملگر ها

عملگر عضویت برای شناسایی بودن یا نبودن یک عبارت در یک شئ استفاده میشود
فرض میکنیم:

```
fruits = ["apple","banana","cherry"]

objects = ["table","door","bed"]
```

عملگر	توضیح	مثال
in	اگر عبارت مورد نظر در شئ وجود داشت مقدار صحیح بر میگرداند	“banana” in fruits
not in	زمانی صحیح است که عبارت در شئ نباشد	“apple” in objects

عملگر هویت برای مقایسه دو شئ استفاده میشود و نه فقط برای مقایسه مقدار آنها
بلکه برای تعیین اینکه آیا این دو شئ یکی هستند نیز استفاده میشود

در واقع این عملگر برسی میکند که آیا دو شئ در حافظه کامپیوتر یک مکان را اشغال میکنند یا نه
یعنی دو شئ ذاتا یکی هستند یا نه

عملگر	توضیح
Is	اگر دو شئ یک حافظه را اشغال کرده باشند مقدار صحیح بر میگرداند
Is not	اگر دو شئ یکی نباشند صحیح بر میگرداند

مثال :

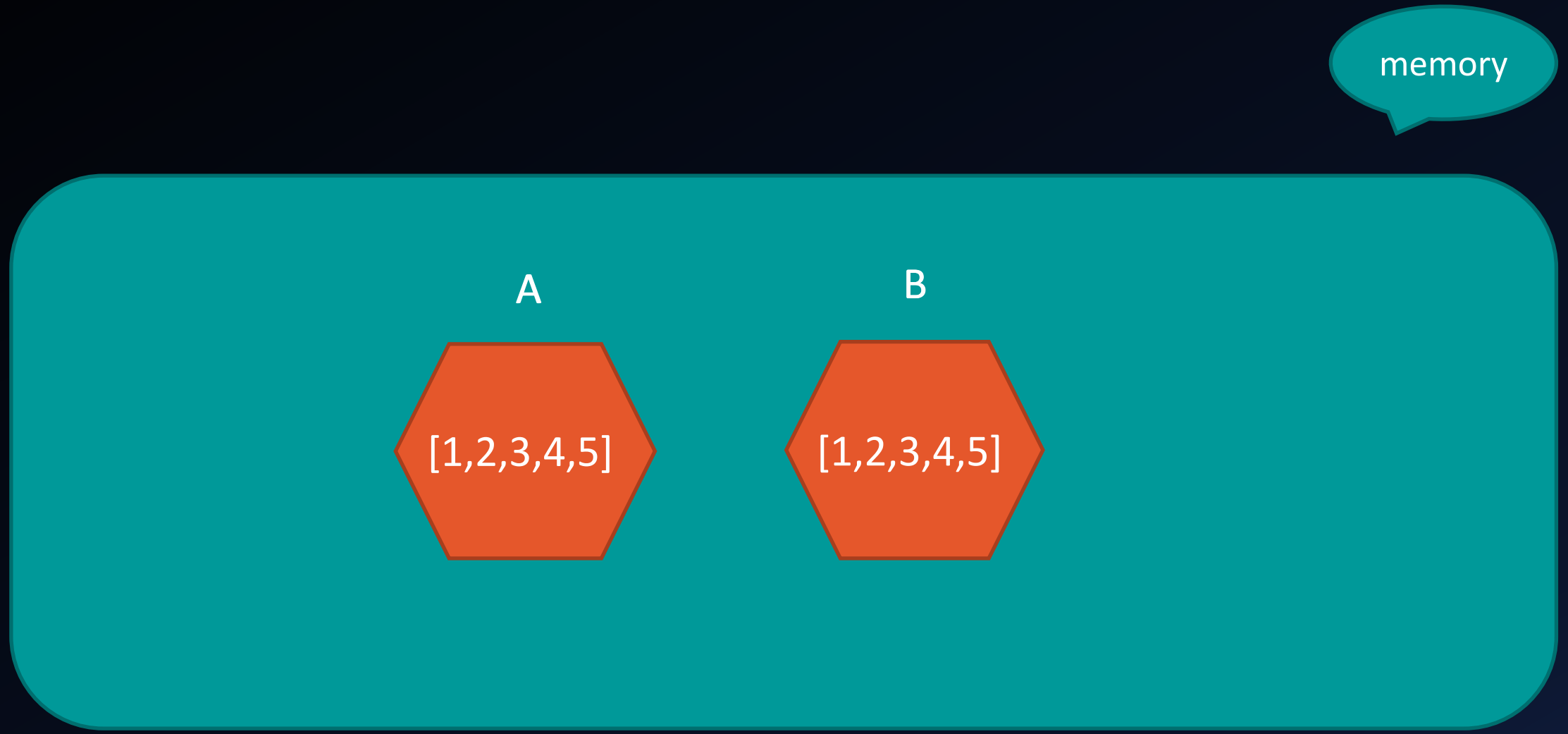
مثال :

A = [1,2,3,4,5] B = [1,2,3,4,5]

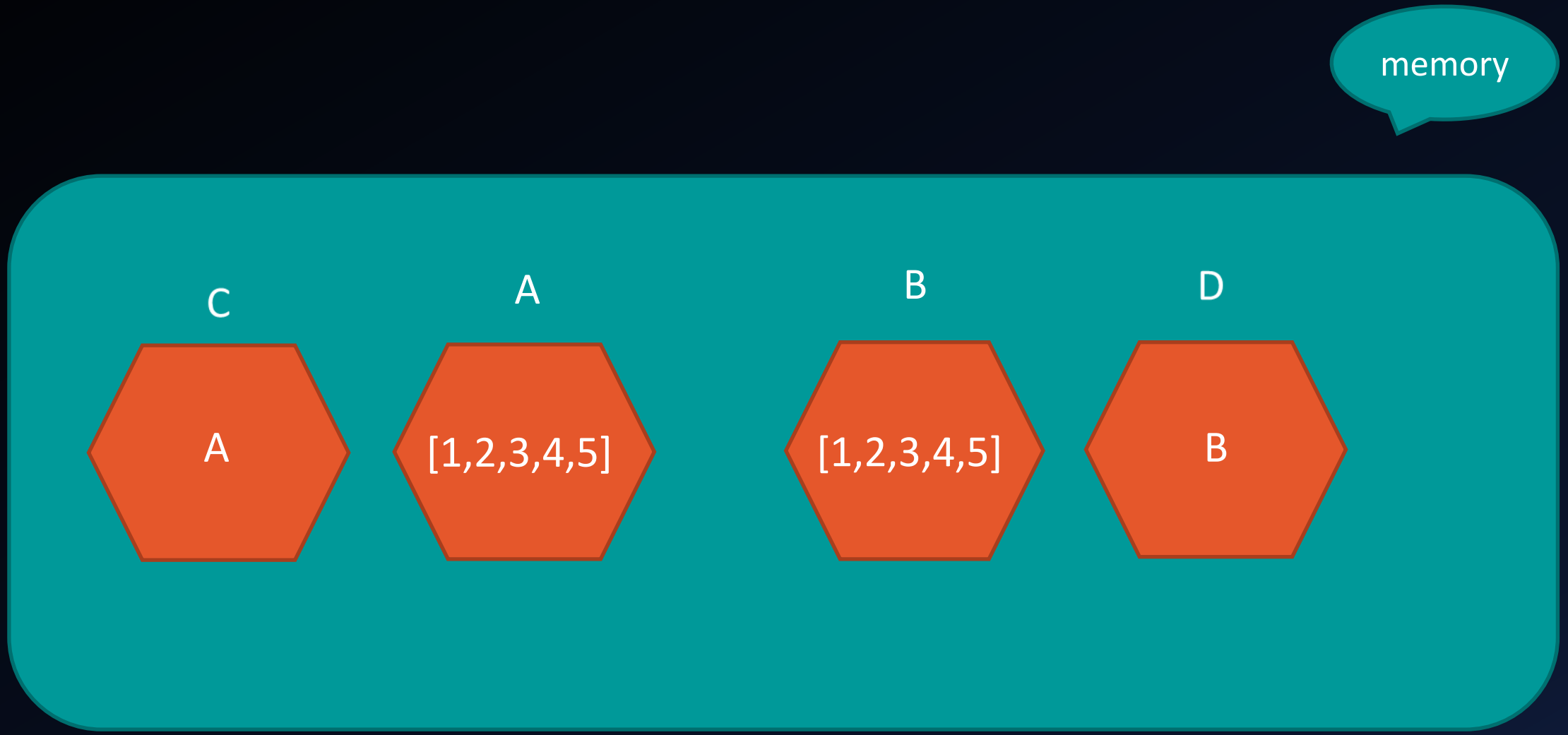
مثال :

C = A

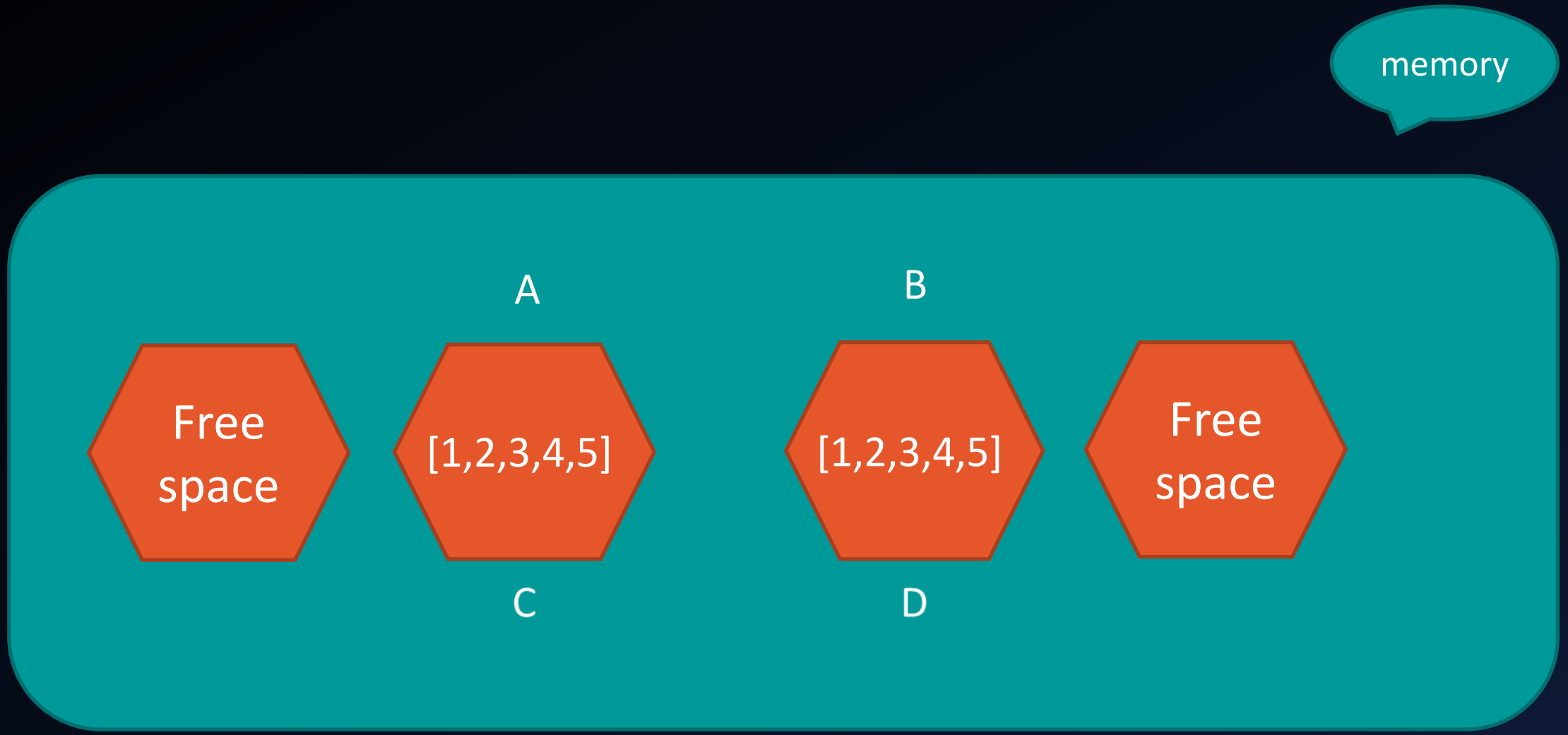
D = B

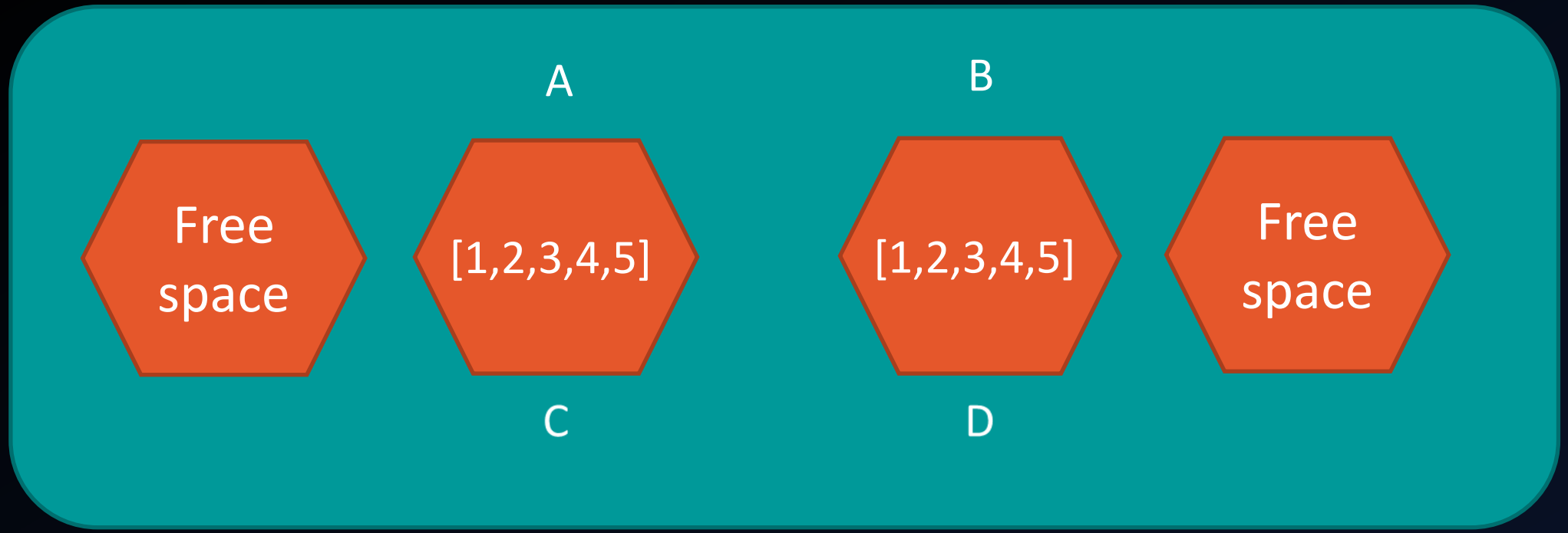


مثال :



مثال :





A is B → False
A is C → True

B is A → False
B is D → True

C is D → False
D is C → False

در واقع (A و C) و (B و D) یک فضا را در حافظه اشغال میکنند و یکی هستند