

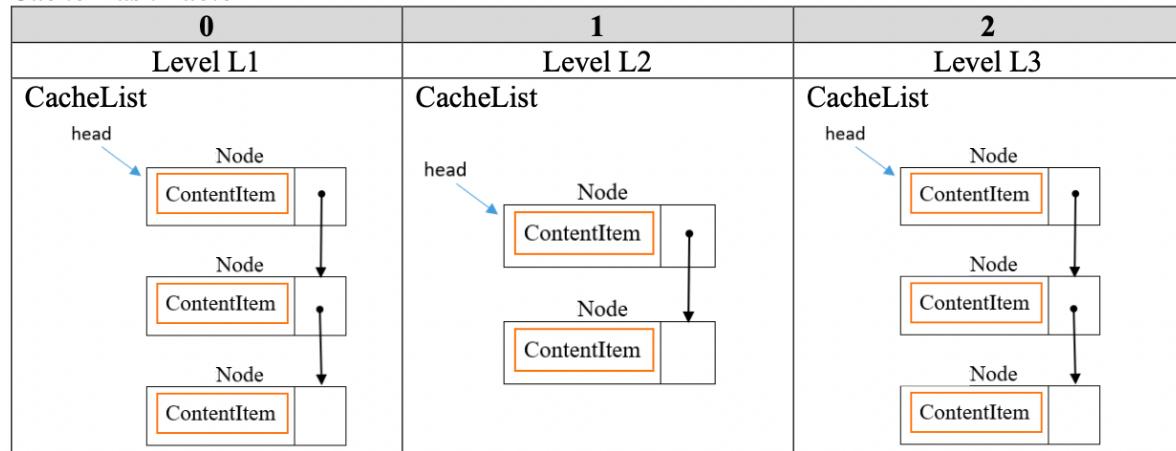
Do not change the string formats given or use the output of `__str__`/`__repr__` in any of the operations you were asked to complete.

The doctest contains <BLANKLINE> in several lines. This line does not need to show up in your code, it simply represents the use of \n in the legible representations of the objects. In your output, it should show just a blank line

In our implementation, we will have three different levels of cache (L1, L2, L3). The overall cache structure will be implemented as a hash table using separate chaining for collision resolution, with each individual level implemented as a linked list. Content is put into different levels based on one of the content's attributes.

Visualization of the Cache, CacheLists, and ContentItems:

Cache Hash Table



The ContentItem class holds a piece of content and is described below. All methods have been implemented for you except `__hash__`. Do not modify the given code.

Attributes

Type	Name	Description
int	cid	Stores the content id.
int	size	Stores the size of the content as a nonnegative integer.
str	header	Information stored by the ContentItem (used for hash function later)
str	content	Information stored by the ContentItem

Special methods

Type	Name	Description
None	<code>__init__(self, cid, size, header, content)</code>	Creates a ContentItem from parameters
int	<code>__hash__(self)</code>	Returns the hash value for this ContentItem
str	<code>__str__(self), __repr__(self)</code>	String representation of this object
bool	<code>__eq__(self, other)</code>	Checks for equality with another object

`__hash__(self)`

Returns the hash value for this ContentItem (used by the Cache class). For this assignment, let the hash value be equal to the sum of every ASCII value in the header, modulo 3. This is the special method for the built-in method `hash(object)`, for example `hash('hello')`. Hint: the [ord\(c\)](#) method could be helpful here.

Output	
int	An integer between 0 and 2 (inclusive), based on the hash function described above

The Node class has been implemented for you and is described below. Do not modify the given code.

Attributes

Type	Name	Description
ContentItem	value	Stores the value of this Node (always a ContentItem in this case)
Node	next	Points to the next Node in the linked list (defaults to None)

Special methods

Type	Name	Description
None	<code>__init__(self, content)</code>	Creates a new Node that holds the given ContentItem
str	<code>__str__(self), __repr__(self)</code>	String representation of this object

The CacheList class describes a single cache level in our hierarchy, implemented as a singly linked list with a reference to the head node. Items are moved to the head every time they are added or used, creating an order in the list from most recently used to least recently used. **READ** the outline for all the methods in this class first, the *put* method should be the last one to be implemented in this class since it relies in the correctness of the other methods.

Attributes

Type	Name	Description
Node	head	Points to the first node in the linked list (defaults to None)
int	maxSize	Maximum size that the CacheList can store
int	remainingSpace	Remaining size that the CacheList can store
int	numItems	The number of items currently in the CacheList

Methods

Type	Name	Description
str	put(self, content, evictionPolicy)	Adds Nodes at the beginning of the list
str	update(self, cid, content)	Updates the content in the list
None	mruEvict(self), lruEvict(self)	Removes the first/last item of the list
str	clear(self)	Removes all items from the list

Special methods

Type	Name	Description
None	<code>__init__(self, size)</code>	Creates a new CacheList with a given maximum size
str	<code>__str__(self), __repr__(self)</code>	String representation of this object
int	<code>__len__(self)</code>	The number of items in the CacheList
bool	<code>__contains__(self, cid)</code>	Determines if a content with cid is in the list

put(self, content, evictionPolicy)

Adds nodes at the beginning of the list and evicts items as necessary to free up space. If the content is larger than the maximum size, do not evict anything. Otherwise, if there is currently not enough space for the content, evict items according to the eviction policy. If the content id exists in the list prior the insertion, new content is not added into the list, but the existing content is moved to the beginning of the list.

Input		
ContentItem	content	The content item to add to the list
str	evictionPolicy	The desired eviction policy (either 'lru' or 'mru')

Output	
str	'INSERTED: <i>contentItem</i> ' if insertion was successful
str	'Insertion not allowed' if content size > maximum size
str	'Content { <i>id</i> } already in cache, insertion not allowed' if <i>id</i> is already present in the list

__contains__(self, cid)

Finds a ContentItem from the list by id, moving the ContentItem to the front of the list if found. This is the special method for the **in** operator to allow the syntax *cid* **in** *object*.

Input	
int	<i>cid</i> The id to search for in the CacheList

Output	
bool	True if the matching ContentItem is in the list, False otherwise

update(self, cid, content)

Updates a ContentItem with a given id in the list. If a match is found, it is moved to the beginning of the list and the old ContentItem is entirely replaced with the new ContentItem. You cannot assume the size of the new content will be the same as the content in the list, thus, you must check that there is enough remaingSpace to perform the update. The update is not completed if the change results on exceeding the maxSize of the list, but the match is moved at the beginning of the list.

Input	
int	<i>cid</i> The id to search for in the CacheList
ContentItem	<i>content</i> The values to update the existing ContentItem with

Output	
str	'UPDATED: <i>contentItem</i> ' if update was successful
str	'Cache miss!' is returned if no match is found or the update exceeds the maxSize

lruEvict(self) / mruEvict(self)

Removes the last (least recently used) or the first (most recently used) item of the list

Output	
None	This function returns nothing.

clear(self)

Removes all items from the list.

Output	
str	'Cleared cache!'

The Cache class describes the overall cache, implemented as a hash table. It contains three CacheLists which actually store the ContentItems. Hash values of 0 correspond with the first CacheList (L1), 1 with L2, and 2 with L3.

0	1	2
Level L1	Level L2	Level L3
CacheList(200)	CacheList(200)	CacheList(200)

All methods in the Cache class will call a corresponding method in the CacheList class. For example, the *insert* method calls the *put* method from the CacheList class.

Do not change the initialization of the CacheList objects in the starter code. **You are not allowed to add any other methods in this class.**

Attributes

Type	Name	Description
list	hierarchy	List with 3 CacheList objects of size 200
int	size	Number of levels in our hierarchy (always set to 3)

Methods

Type	Name	Description
str	insert(self, content, evictionPolicy)	Adds an item into the proper cache list
(various)	updateContent(self, content)	Updates an item from the proper cache list
str	clear(self)	Clears all CacheLists in the hierarchy.

Special methods

Type	Name	Description
None	<code>__init__(self)</code>	Creates a new Cache with (3) CacheLists of size 200
str	<code>__str__(self), __repr__(self)</code>	String representation of this object
(various)	<code>__getitem__(self, content)</code>	Gets an item from the proper cache list

insert(self, content, evictionPolicy)

Inserts a ContentItem into the proper CacheList. After using the hash function to determine which CacheList the content should go into, call that CacheList's put method to add the content.

Input		
ContentItem	content	The content item to add to the list
str	evictionPolicy	The desired eviction policy (either 'lru' or 'mru')

Output	
str	(Return the output from the put method call)

__getitem__(self, content)

Gets a ContentItem from a CacheList. After using the hash function to determine which CacheList the content should exist in, use call that CacheList's `in` operator to return the content. This is the special method to support the syntax `object[content]`

Input		
ContentItem	content	The content item to retrieve

Output	
ContentItem	The matching ContentItem is returned if it's a cache hit (item was found)
str	'Cache miss!' is returned if it's a cache miss (item was not found)

updateContent(self, content)

Updates a ContentItem. After using the hash function to determine which CacheList the content would be in, call that CacheList's update method to update the content.

Input		
ContentItem	content	The content item to update

Output	
ContentItem	The updated ContentItem is returned if it's a cache hit (item was found)
str	'Cache miss!' is returned if it's a cache miss (item was not found)

clear(self)

Clears all the lists in the hierarchy. This method is already implemented for you.

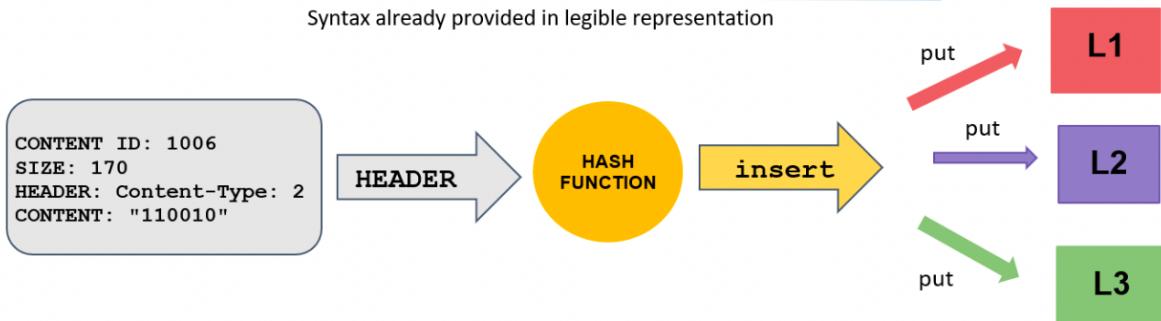
Output	
str	'Cache cleared!'

```

>>> cache = Cache()
>>> content1 = ContentItem(1006, 170, "Content-Type: 2", "110010")
>>> cache.insert(content1, 'mru')
'INSERTED: CONTENT ID: 1006 SIZE: 170 HEADER: Content-Type: 2 CONTENT: 110010'

```

Syntax already provided in legible representation



```

>>> cache = Cache()
>>> content1 = ContentItem(1006, 170, "Content-Type: 2", "110010")
>>> cache.insert(content1, 'mru')
'INSERTED: CONTENT ID: 1006 SIZE: 170 HEADER: Content-Type: 2 CONTENT: 110010'
>>> cache[content1]
CONTENT ID: 1006 SIZE: 170 HEADER: Content-Type: 2 CONTENT: 110010

```

Syntax already provided in legible representation, just return the ContentItem object



```

>>> cache = Cache()
>>> content1 = ContentItem(1006, 170, "Content-Type: 2", "110010")
>>> cache.insert(content1, 'mru')
'INSERTED: CONTENT ID: 1006 SIZE: 170 HEADER: Content-Type: 2 CONTENT: 110010'
>>> content5 = ContentItem(1006, 170, "Content-Type: 2", "11111111111111")
>>> cache.updateContent(content5)
'UPDATED: CONTENT ID: 1006 SIZE: 170 HEADER: Content-Type: 2 CONTENT: 11111111111111'

```

