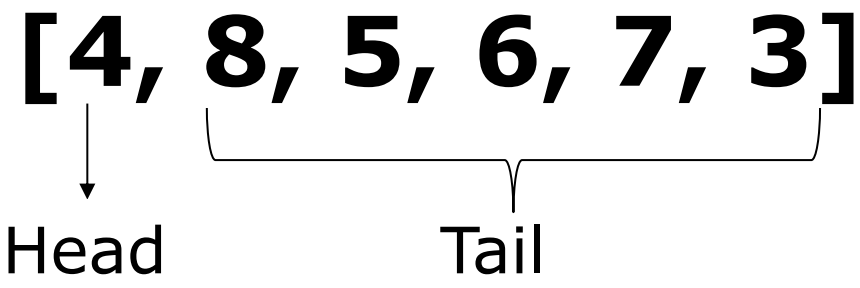
What are Lists

The next Data Structure that we learn is called a **list**.

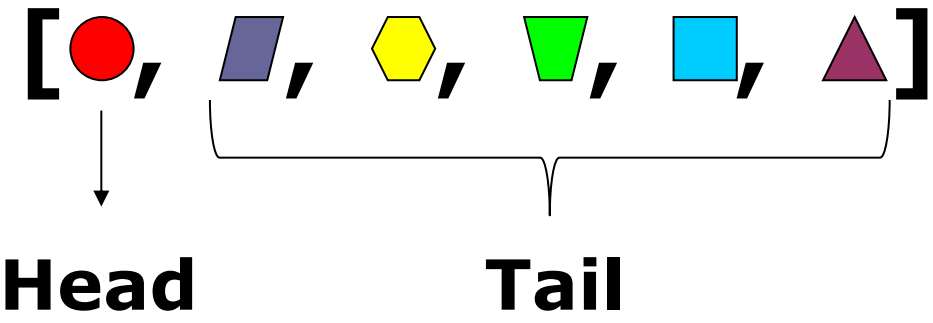


Operations:

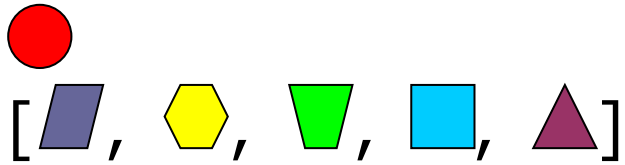
There are many operations on lists. The basic ones are

- Constructing a list from a head and a tail.
- Obtaining the head of a list.
- Obtaining the tail of a list.

Head and Tail of a List



Head:
Tail:



Head is an element.
Tail is a list.

Head and Tail of a List

['a', 'd', 'e', 'b', 's', 'a', 'e']

Head: 'a'
Tail: ['d', 'e', 'b', 's', 'a', 'e']

Head is an element.
Tail is a list.

Head and Tail of a List

[4, 5, 8, 9, 7, 6, 3]

Head: 4
Tail: [5, 8, 9, 7, 6, 3]

Head is an element.
Tail is a list.

Head and Tail of a List

[4]

Head: 4
Tail: []

Head is an element.
Tail is a list.

Head and Tail of the Empty List

[]

Head: -
Tail: -

Empty List has no head.
Empty List has no tail.
(*In fact, Empty List is usually seen as a constant.*)

Head and Tail of a List

[[1,2], [3,4,8,9], [5,6,7]]

Head: [1,2]
Tail: [[3,4,8,9], [5,6,7]]

Head is an element.
Tail is a list.

Head and Tail of a List

['a', [3,4,8,9], , 23]

Head: 'a'
Tail: [[3,4,8,9], , 23]

Head is an element.
Tail is a list.

Head and Tail of a List

[[], [2,3], [1]]

Head: []
Tail: [[2,3], [1]]

Head is an element.
Tail is a list.

Head and Tail of a List

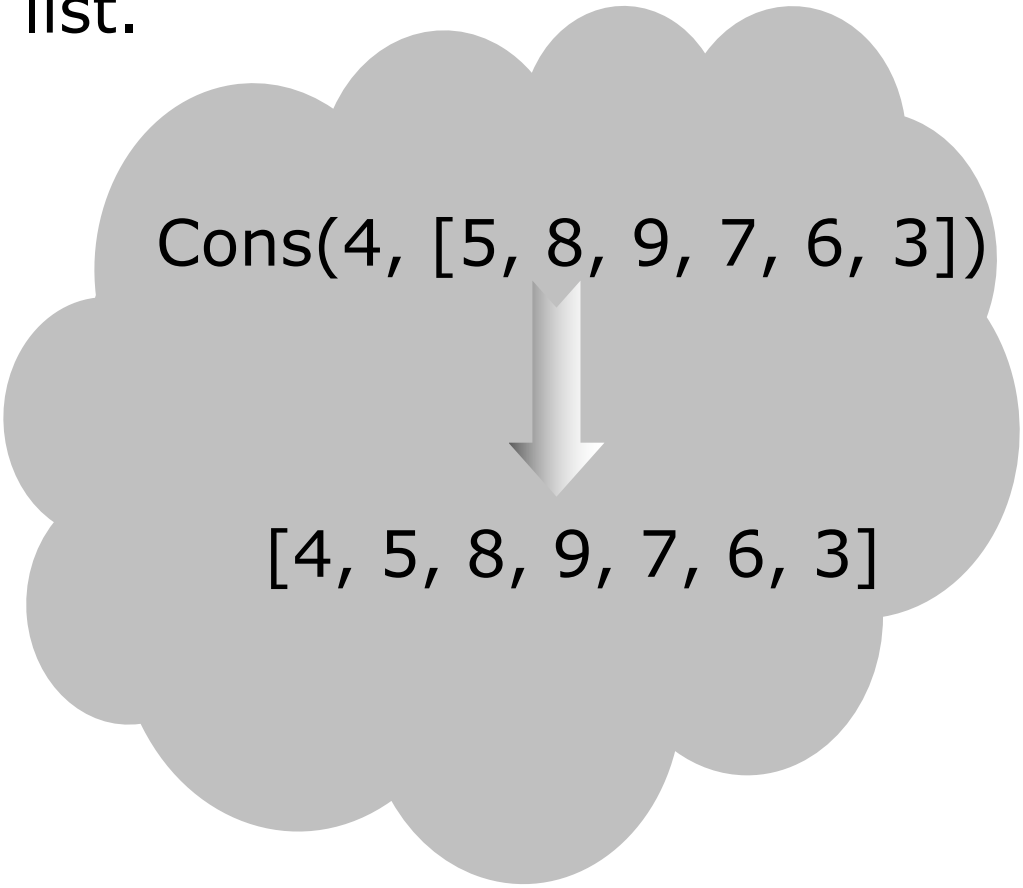
[[]]

Head: []
Tail: []

Head is an element.
Tail is a list.

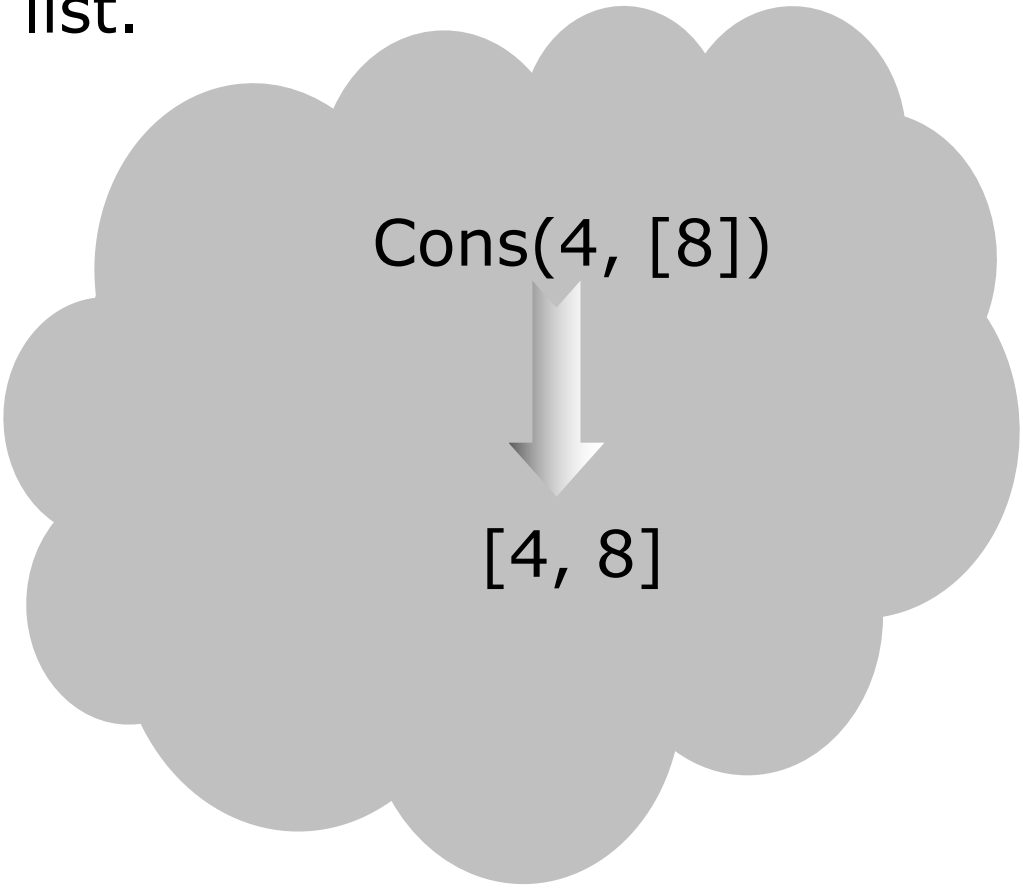
List Operations

The **list construction** operation constructs a list from a head and a list.



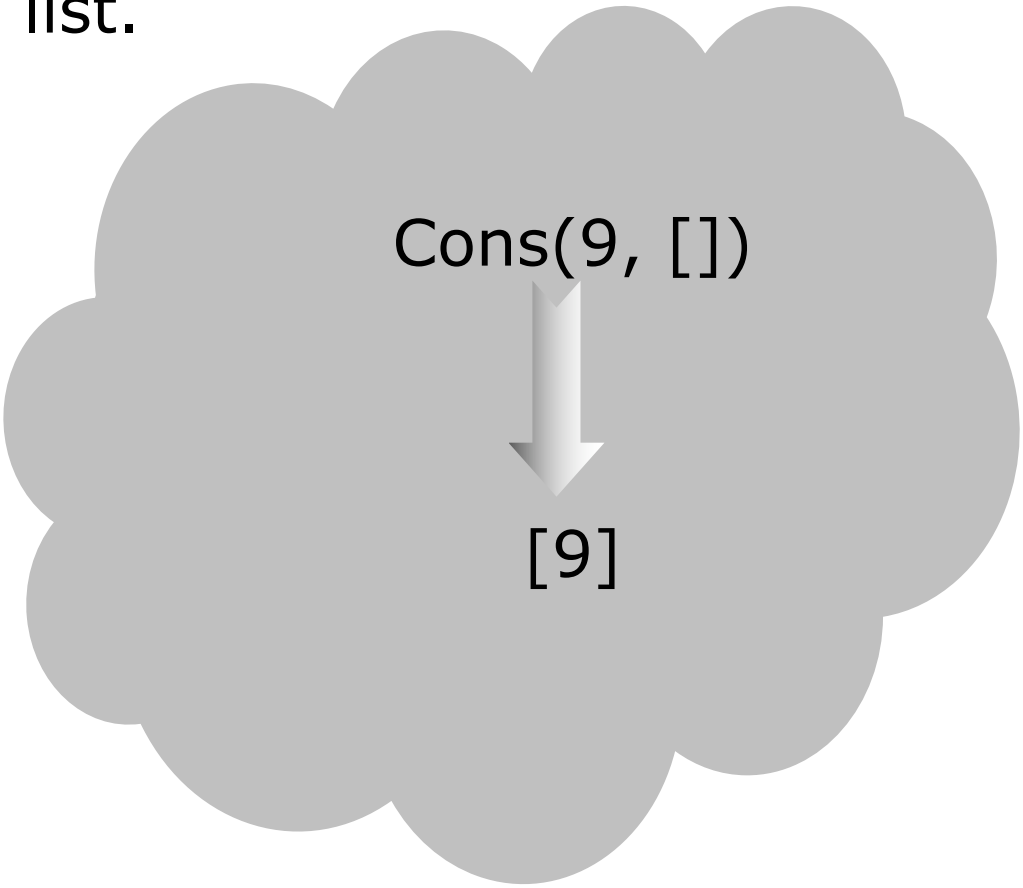
List Operations

The **list construction** operation constructs a list from a head and a list.



List Operations

The **list construction** operation constructs a list from a head and a list.



List Operations

The **list construction** operation constructs a list from a head and a list.

Cons(7, Cons(8, Cons(9, [])))

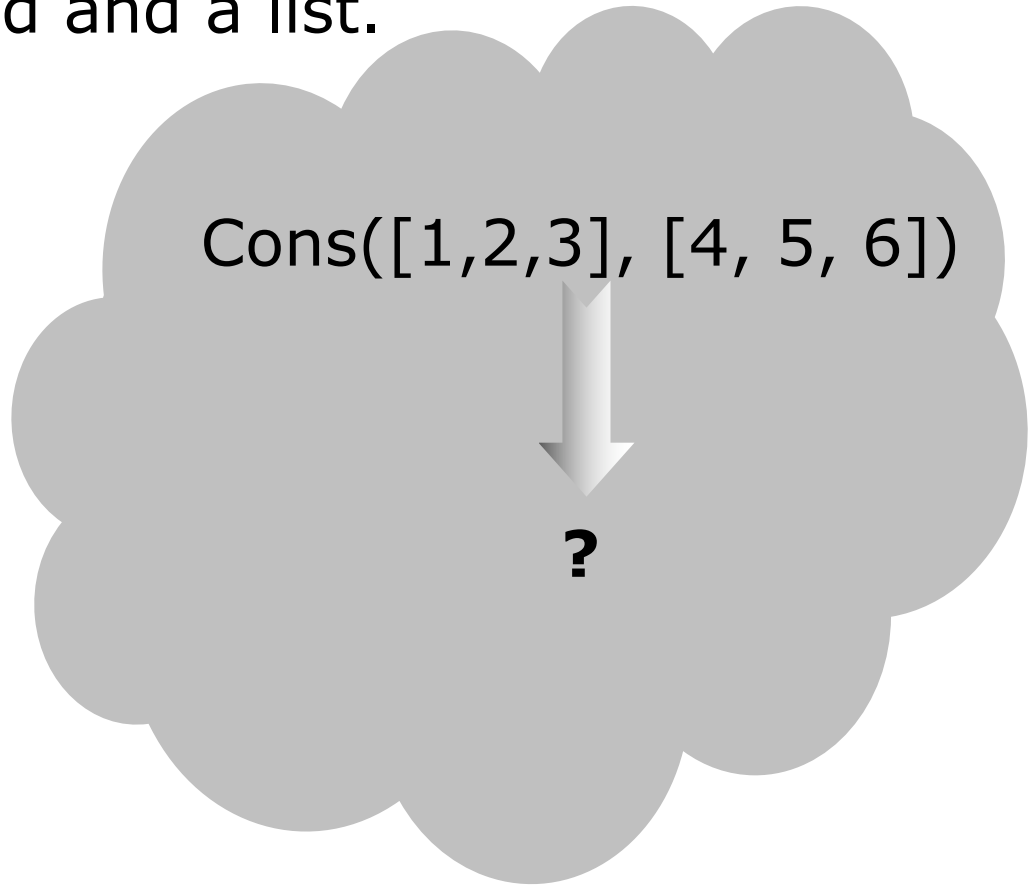
Cons(7, Cons(8, **[9]**))

Cons(7, **[8, 9]**)

[7, 8, 9]

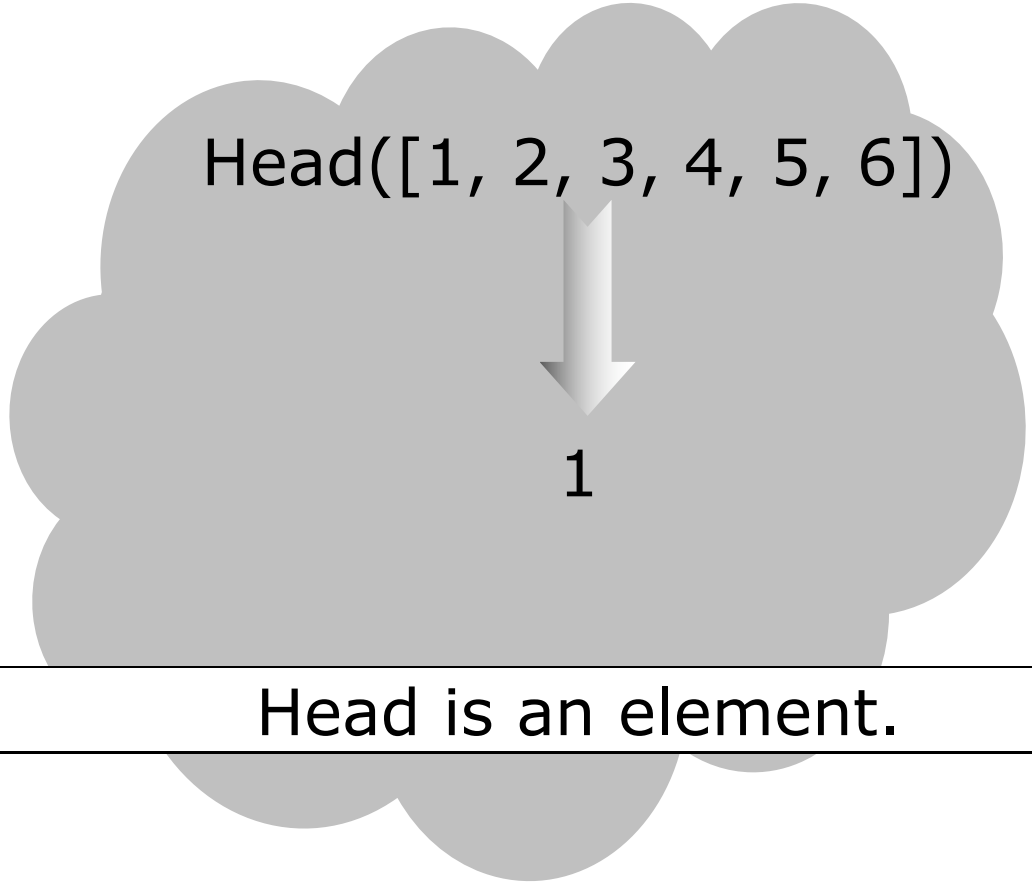
List Operations

The **list construction** operation constructs a new list from a head and a list.



List Operations

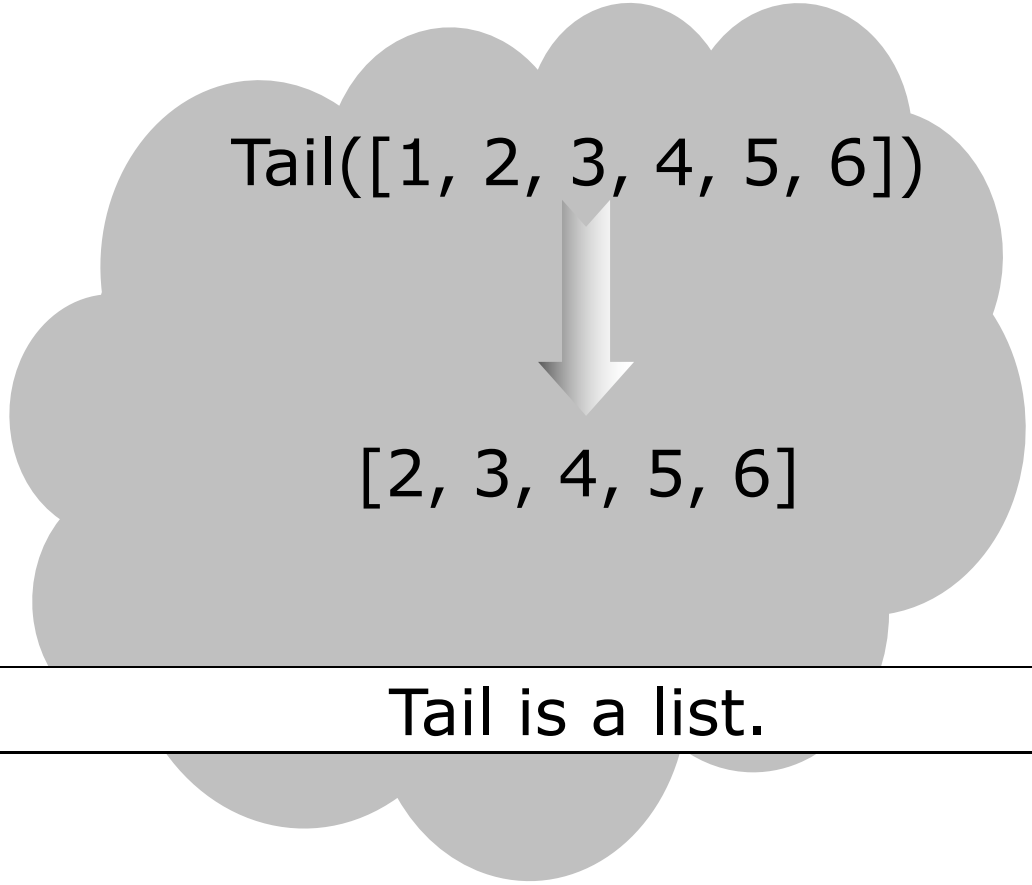
The **list head** operation returns the head of a list.



Head is an element.

List Operations

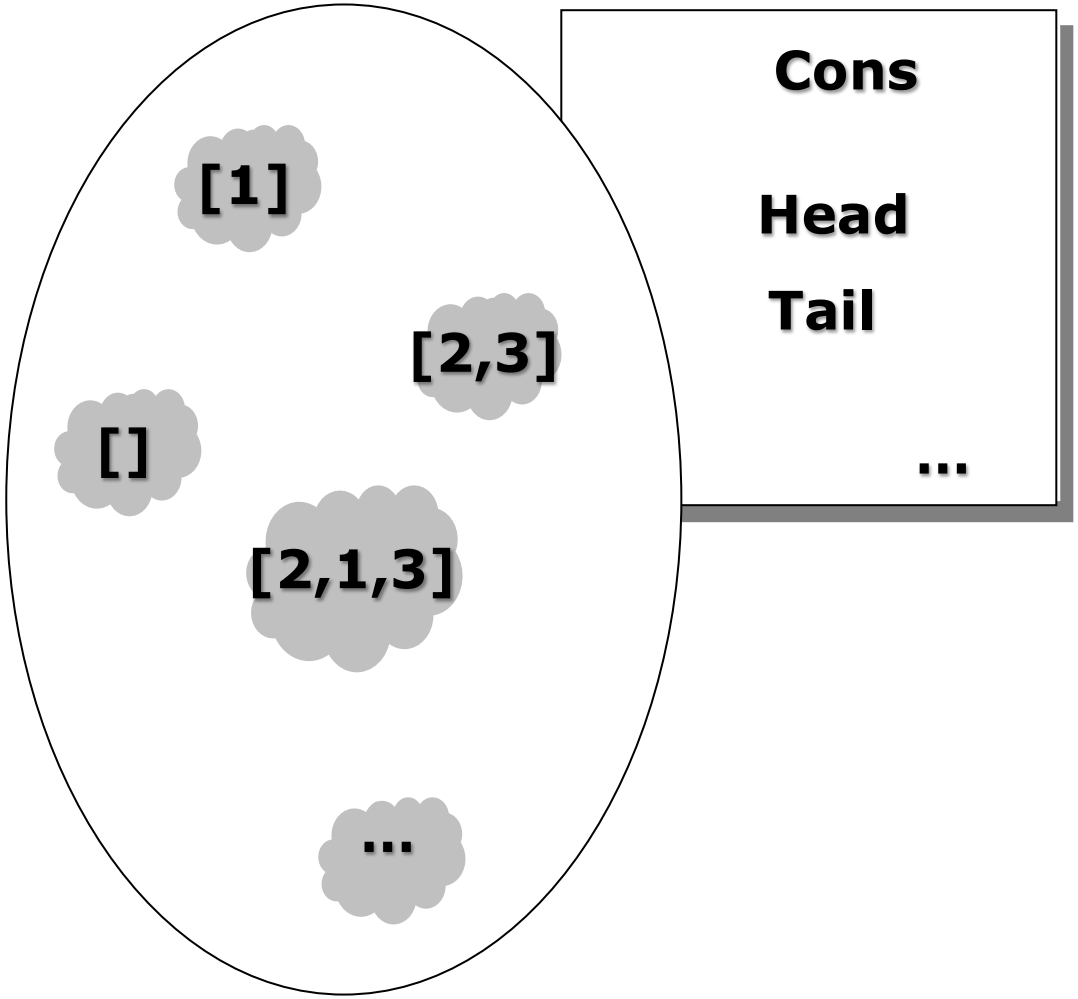
The **list tail** operation returns the tail of a list.



Tail is a list.

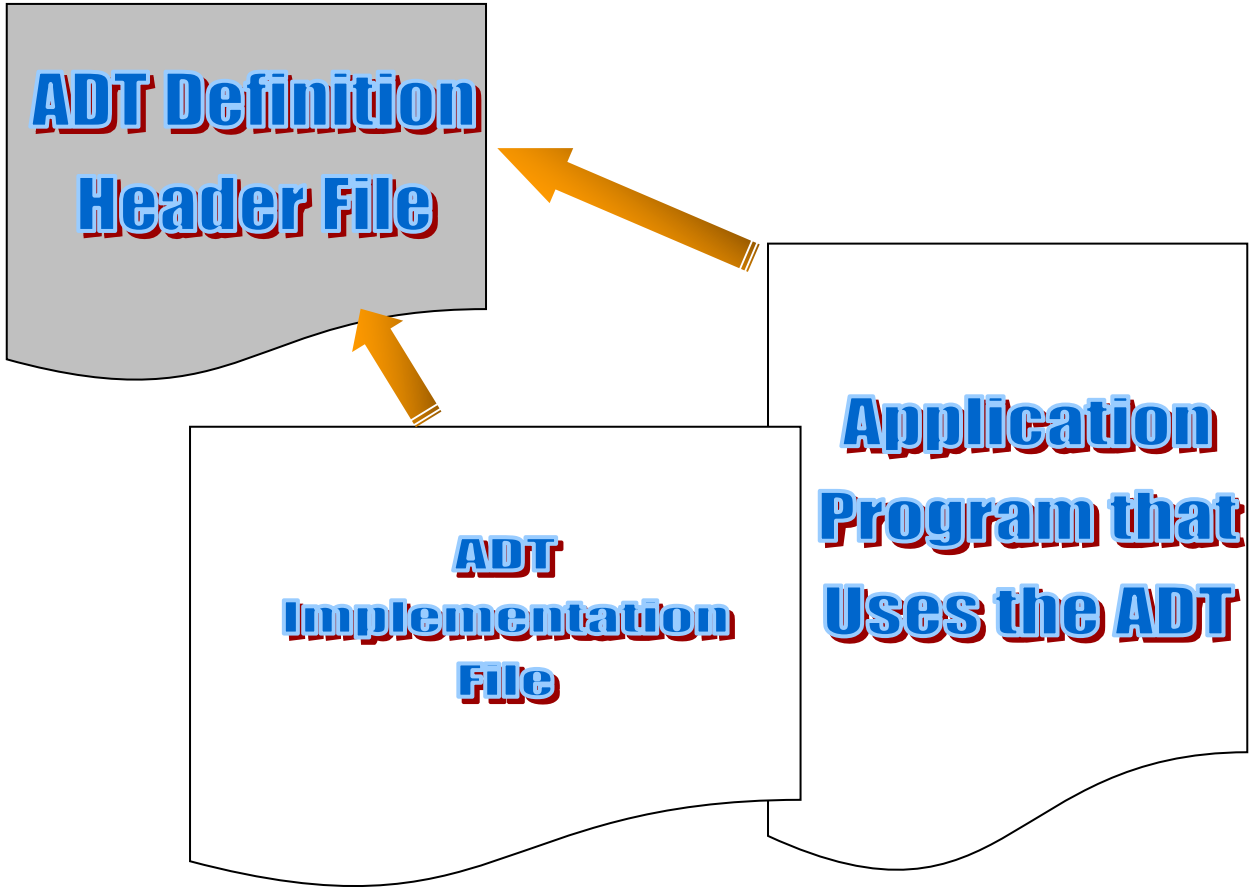
Defining a List ADT

As before, we see that LIST is an Abstract Data Type.



Defining the Type listADT

We now proceed to do what we should do: to write the file **list.h**.



*

```
/*
 * File: list.h
 */

typedef struct listCDT *listADT;

typedef int listElementT;

listADT EmptyList(void);
listADT Cons(listElementT, listADT);
listElementT Head(listADT);
listADT Tail(listADT);
int ListIsEmpty(listADT);
```

Notes

- **typedef struct listCDT *listADT;**
  As before, this declaration is a must.

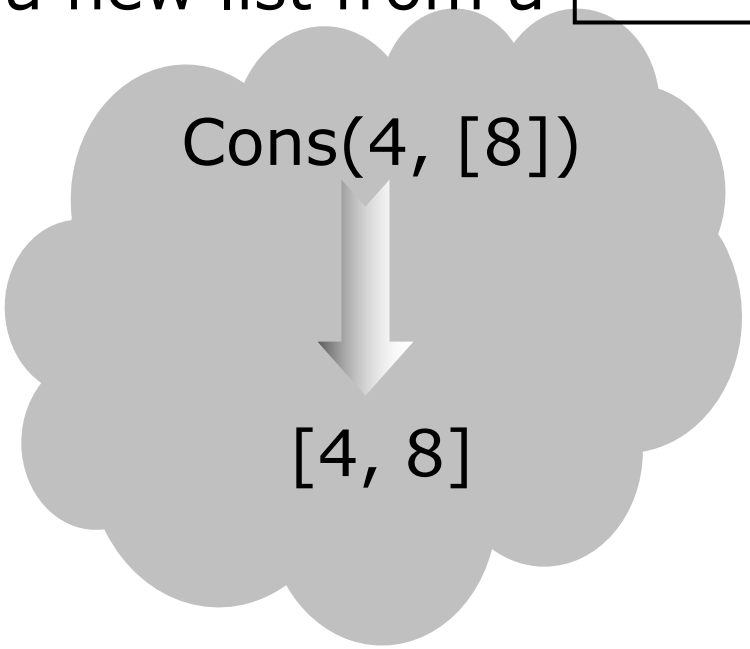
- **typedef int listElementT;**
  In version 1.0, we only consider **list of integers**.

  []
  [1, 2, 3]
  [8]
  [4, 8, 7, 6]

```
/*
 * File: list.h
 */

typedef struct listCDT *listADT;

typedef int listElementT;

listADT EmptyList(void);
listADT Cons(listElementT, listADT);
listElementT Head(listADT);
listADT Tail(listADT);
int ListIsEmpty(listADT);
```

- **listADT EmptyList(void);**
  This returns a new empty list.

- **listADT Cons(listElementT, listADT);**
  This constructs a new list from a head and a tail.




```
/*
 * File: list.h
 */

typedef struct listCDT *listADT;

typedef int listElementT;

listADT EmptyList(void);
listADT Cons(listElementT, listADT);
listElementT Head(listADT);
listADT Tail(listADT);
int ListIsEmpty(listADT);
```

- **listElementT Head(listADT);**
  This returns the head of a list.

Head([1, 2, 3, 4, 5, 6])

1

```
/*
 * File: list.h
 */

typedef struct listCDT *listADT;

typedef int listElementT;

listADT EmptyList(void);
listADT Cons(listElementT, listADT);
listElementT Head(listADT);
listADT Tail(listADT);
int ListIsEmpty(listADT);
```

- **listADT Tail(listADT);**
  This returns the tail of a list.

Tail([1, 2, 3, 4, 5, 6])

[2, 3, 4, 5, 6]

- **int ListIsEmpty(listADT);**
  This returns 1 if the listADT argument is empty, and 0 otherwise.

```
/*
 * File: list.h
 */

typedef struct listCDT *listADT;

typedef int listElementT;

listADT EmptyList(void);
listADT Cons(listElementT, listADT);
listElementT Head(listADT);
listADT Tail(listADT);
int ListIsEmpty(listADT);
```

Example

Assign the empty list ([]) to a list L1.

```
listADT L1;

L1 = EmptyList();
```

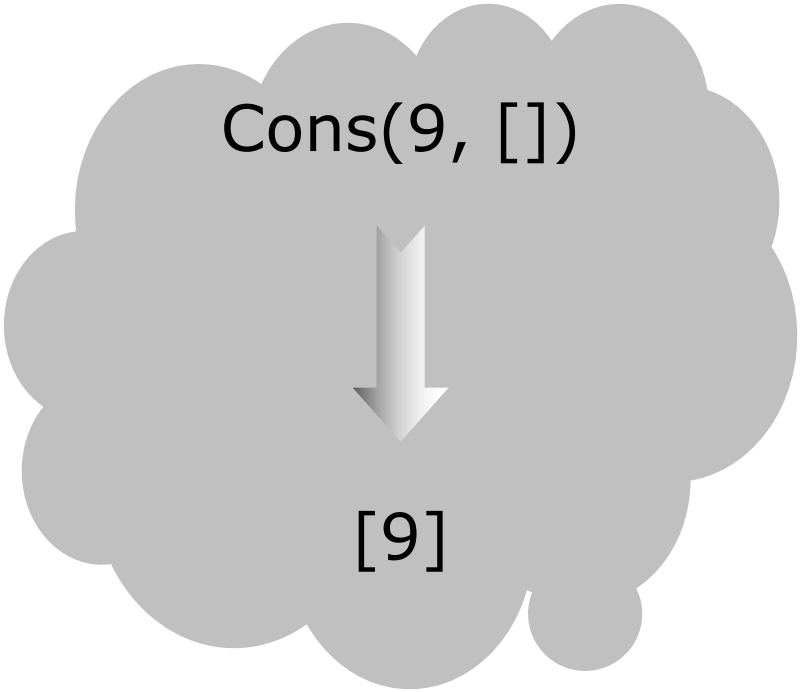
Question:

- What is Head(L1)?
- What is Tail(L1)?
- What is ListIsEmpty(L1)?

Example

Assign the list [9] to a list L1.

```
listADT L1, L2;

L2 = EmptyList();
L1 = Cons(9, L2);
```



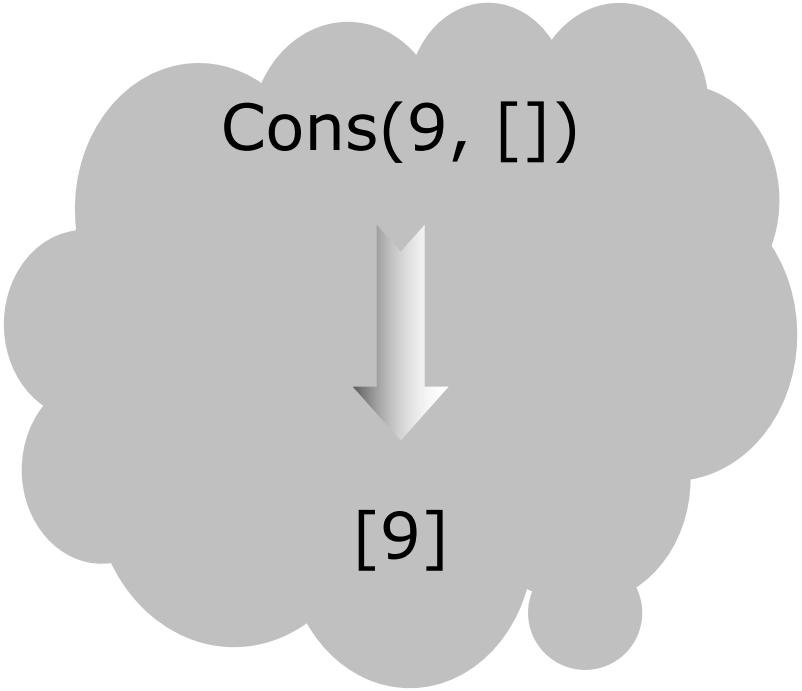
Questions:

- What is Head(L1)?
- What is Tail(L1)?
- What is ListIsEmpty(L1)?

Example

Assign the list [9] to a list L1.

```
listADT L1, L2;

L1 = Cons(9, EmptyList());
```



Questions:

- What is Head(L1)?
- What is Tail(L1)?
- What is ListIsEmpty(L1)?

Example

How about this.

```
listADT L1;

L1 = EmptyList();
L1 = Cons(10, L1);
```

Questions:

- What is Head(L1)?
- What is Tail(L1)?
- What is ListIsEmpty(L1)?

Example

Assign the list [5, 6, 7] to a list L1.

```
listADT L1, L2, L3;

L3 = Cons(7, EmptyList());
L2 = Cons(6, L3);
L1 = Cons(5, L2);
```

[ ]

[7]

[6,7]

[5,6,7]

Questions:

- What is Head(L1)?
- What is Tail(L1)?
- What is ListIsEmpty(L1)?

Example

Assign the list [5, 6, 7] to a list L1.

```
listADT L1, L2;

L1 = Cons(7, EmptyList());
L2 = Cons(6, L1);
L1 = Cons(5, L2);
```

Questions:

- What is L1?
- What is L2?

Example

Assign the list [5, 6, 7] to a list L1.

```
listADT L1, L2;

L2 = Cons(6, Cons(7, EmptyList()));
L1 = Cons(5, L2);
L2 = EmptyList();
```

Questions:

- What is L1?
- What is L2?

Example

Assign the list [5, 6, 7] to a list L1.

```
listADT L1, L2, L3;

L1 = Cons(5, Cons(6, Cons(7, EmptyList())));
```

Questions:

- What is Head(L1)?
- What is Tail(L1)?
- What is ListIsEmpty(L1)?

Example

Assign the list [5, 6, 7] to a list L1.

```
listADT L1, L2, L3;

L1 = Cons(5, Cons(6, Cons(7, EmptyList())));
```

Questions:

- What is Head(Tail(L1))?
- What is Tail(Tail(L1))?
- What is ListIsEmpty(Tail(Tail(L1)))?
- What is ListIsEmpty(Tail(Tail(Tail(L1))))?