

Data Examples

Announcements

Examples: Objects

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
```

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:  
    greeting = 'Sir'
```

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
```

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
```


Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting
```

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
```

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
```

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
```

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

```
>>> Worker().work()
'Sir, I work'
>>> jack
Peon
>>> jack.work()
'Maam, I work'
>>> john.work()
Peon, I work
'I gather wealth'
>>> john.elf.work(john)
'Peon, I work'
```

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting
```

```
class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
```

```
jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

```
>>> Worker().work()
```

```
>>> jack
```

```
>>> jack.work()
```

```
>>> john.work()
```

```
>>> john.elf.work(john)
```

<class Worker>

greeting: 'Sir'

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting
```

```
class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
```

```
jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

```
>>> Worker().work()
```

```
>>> jack
```

```
>>> jack.work()
```

```
>>> john.work()
```

```
>>> john.elf.work(john)
```

<class Worker>

greeting: 'Sir'

<class Bourgeoisie>

greeting: 'Peon'

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting
```

```
class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
```

```
jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

```
>>> Worker().work()
```

```
>>> jack
```

```
>>> jack.work()
```

```
>>> john.work()
```

```
>>> john.elf.work(john)
```

<class Worker>

greeting: 'Sir'

<class Bourgeoisie>

greeting: 'Peon'

jack <Worker>

elf: _____

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting
```

```
class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
```

```
jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

```
>>> Worker().work()
```

```
>>> jack
```

```
>>> jack.work()
```

```
>>> john.work()
```

```
>>> john.elf.work(john)
```

<class Worker>

greeting: 'Sir'

<class Bourgeoisie>

greeting: 'Peon'

jack <Worker>

elf: _____

john <Bourgeoisie>

elf: _____

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting
```

```
class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
```

```
jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

```
>>> Worker().work()
```

```
>>> jack
```

```
>>> jack.work()
```

```
>>> john.work()
```

```
>>> john.elf.work(john)
```


<class Worker>

greeting: 'Sir'

<class Bourgeoisie>

greeting: 'Peon'

jack <Worker>

elf: 
greeting: 'Maam'

john <Bourgeoisie>

elf: 

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting
```

```
class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
```

```
jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

```
>>> Worker().work()
```

```
>>> jack
```

```
>>> jack.work()
```

```
>>> john.work()
```

```
>>> john.elf.work(john)
```


<class Worker>

greeting: 'Sir'

<class Bourgeoisie>

greeting: 'Peon'

jack <Worker>

elf: 
greeting: 'Maam'

john <Bourgeoisie>

elf: 

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting
```

```
class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
```

```
jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

```
>>> Worker().work()
'Sir, I work'
```

```
>>> jack
```

```
>>> jack.work()
```

```
>>> john.work()
```

```
>>> john.elf.work(john)
```


<class Worker>

greeting: 'Sir'

<class Bourgeoisie>

greeting: 'Peon'

jack <Worker>

elf: 
greeting: 'Maam'

john <Bourgeoisie>

elf: 

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting
```

```
class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
```

```
jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

```
>>> Worker().work()
'Sir, I work'
```

```
>>> jack
```

```
>>> jack.work()
```

```
>>> john.work()
```

```
>>> john.elf.work(john)
```


<class Worker>

greeting: 'Sir'

<class Bourgeoisie>

greeting: 'Peon'

jack <Worker>

elf: 
greeting: 'Maam'

john <Bourgeoisie>

elf: 

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting
```

```
class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
```

```
jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

```
>>> Worker().work()
'Sir, I work'
```

```
>>> jack
Peon
```

```
>>> jack.work()
```

```
>>> john.work()
```

```
>>> john.elf.work(john)
```


<class Worker>

greeting: 'Sir'

<class Bourgeoisie>

greeting: 'Peon'

jack <Worker>

elf: 
greeting: 'Maam'

john <Bourgeoisie>

elf: 

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting
```

```
class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
```

```
jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

```
>>> Worker().work()
'Sir, I work'
```

```
>>> jack
Peon
```

```
>>> jack.work()
```

```
>>> john.work()
```

```
>>> john.elf.work(john)
```


<class Worker>

greeting: 'Sir'

<class Bourgeoisie>

greeting: 'Peon'

jack <Worker>

elf: 
greeting: 'Maam'

john <Bourgeoisie>

elf: 

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting
```

```
class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
```

```
jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

```
>>> Worker().work()
'Sir, I work'
```

```
>>> jack
Peon
```

```
>>> jack.work()
'Maam, I work'
```

```
>>> john.work()
```

```
>>> john.elf.work(john)
```

<class Worker>

greeting: 'Sir'

<class Bourgeoisie>

greeting: 'Peon'

jack <Worker>

elf: 
greeting: 'Maam'

john <Bourgeoisie>

elf: 

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting
```

```
class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
```

```
jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

```
>>> Worker().work()
'Sir, I work'
```

```
>>> jack
Peon
```

```
>>> jack.work()
'Maam, I work'
```

```
>>> john.work()
```

```
>>> john.elf.work(john)
```

<class Worker>

greeting: 'Sir'

<class Bourgeoisie>

greeting: 'Peon'

jack <Worker>

elf: 
greeting: 'Maam'

john <Bourgeoisie>

elf: 

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting
```

```
class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
```

```
jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

```
>>> Worker().work()
'Sir, I work'
```

```
>>> jack
Peon
```

```
>>> jack.work()
'Maam, I work'
```

```
>>> john.work()
Peon, I work
'I gather wealth'
```

```
>>> john.elf.work(john)
```


<class Worker>

greeting: 'Sir'

<class Bourgeoisie>

greeting: 'Peon'

jack <Worker>

elf: 
greeting: 'Maam'

john <Bourgeoisie>

elf: 

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting
```

```
class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
```

```
jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

```
>>> Worker().work()
'Sir, I work'
```

```
>>> jack
Peon
```

```
>>> jack.work()
'Maam, I work'
```

```
>>> john.work()
Peon, I work
'I gather wealth'
```

```
>>> john.elf.work(john)
```


<class Worker>

greeting: 'Sir'

<class Bourgeoisie>

greeting: 'Peon'

jack <Worker>

elf: 
greeting: 'Maam'

john <Bourgeoisie>

elf: 

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting
```

```
class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
```

```
jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

```
>>> Worker().work()
'Sir, I work'
```

```
>>> jack
Peon
```

```
>>> jack.work()
'Maam, I work'
```

```
>>> john.work()
Peon, I work
'I gather wealth'
```

```
>>> john.elf.work(john)
'Peon, I work'
```

<class Worker>

greeting: 'Sir'

<class Bourgeoisie>

greeting: 'Peon'

jack <Worker>

elf: 
greeting: 'Maam'

john <Bourgeoisie>

elf: 

Examples: Iterables & Iterators

Using Built-In Functions & Comprehensions

Using Built-In Functions & Comprehensions

What are the indices of all elements in a list `s` that have the smallest absolute value?

Using Built-In Functions & Comprehensions

What are the indices of all elements in a list `s` that have the smallest absolute value?

```
[-4, -3, -2, 3, 2, 4]
```


Using Built-In Functions & Comprehensions

What are the indices of all elements in a list `s` that have the smallest absolute value?

```
[-4, -3, -2, 3, 2, 4]  
 0  1  2  3  4  5
```


Using Built-In Functions & Comprehensions

What are the indices of all elements in a list `s` that have the smallest absolute value?

`[-4, -3, -2, 3, 2, 4]`
`0 1 2 3 4 5`  `[2, 4]`



Using Built-In Functions & Comprehensions

What are the indices of all elements in a list `s` that have the smallest absolute value?

`[-4, -3, -2, 3, 2, 4]`
`0 1 2 3 4 5`  `[2, 4]` `[1, 2, 3, 4, 5]`



Using Built-In Functions & Comprehensions

What are the indices of all elements in a list `s` that have the smallest absolute value?

`[-4, -3, -2, 3, 2, 4]`
`0 1 2 3 4 5`  `[2, 4]` `[1, 2, 3, 4, 5]`  `[0]`

Using Built-In Functions & Comprehensions



What are the indices of all elements in a list `s` that have the smallest absolute value?

`[-4, -3, -2, 3, 2, 4]`
`0 1 2 3 4 5`  `[2, 4]` `[1, 2, 3, 4, 5]`  `[0]`

What's the largest sum of two adjacent elements in a list `s`? (Assume `len(s) > 1`)

Using Built-In Functions & Comprehensions

What are the indices of all elements in a list `s` that have the smallest absolute value?



`[-4, -3, -2, 3, 2, 4]`
`0 1 2 3 4 5`  `[2, 4]` `[1, 2, 3, 4, 5]`  `[0]`

What's the largest sum of two adjacent elements in a list `s`? (Assume `len(s) > 1`)


`[-4, -3, -2, 3, 2, 4]`

Using Built-In Functions & Comprehensions

What are the indices of all elements in a list `s` that have the smallest absolute value?



`[-4, -3, -2, 3, 2, 4]`
`0 1 2 3 4 5`  `[2, 4]` `[1, 2, 3, 4, 5]`  `[0]`

What's the largest sum of two adjacent elements in a list `s`? (Assume `len(s) > 1`)

`[-4, -3, -2, 3, 2, 4]`  `6`

Using Built-In Functions & Comprehensions

What are the indices of all elements in a list `s` that have the smallest absolute value?



`[-4, -3, -2, 3, 2, 4]`
`0 1 2 3 4 5`  `[2, 4]` `[1, 2, 3, 4, 5]`  `[0]`

What's the largest sum of two adjacent elements in a list `s`? (Assume `len(s) > 1`)

`[-4, -3, -2, 3, 2, 4]`  `6` `[-4, 3, -2, -3, 2, -4]`

Using Built-In Functions & Comprehensions

What are the indices of all elements in a list `s` that have the smallest absolute value?



`[-4, -3, -2, 3, 2, 4]`
`0 1 2 3 4 5`  `[2, 4]` `[1, 2, 3, 4, 5]`  `[0]`

What's the largest sum of two adjacent elements in a list `s`? (Assume `len(s) > 1`)



`[-4, -3, -2, 3, 2, 4]`  `6` `[-4, 3, -2, -3, 2, -4]`  `1`

Using Built-In Functions & Comprehensions

What are the indices of all elements in a list `s` that have the smallest absolute value?

`[-4, -3, -2, 3, 2, 4]`
`0 1 2 3 4 5`  `[2, 4]` `[1, 2, 3, 4, 5]`  `[0]`



What's the largest sum of two adjacent elements in a list `s`? (Assume `len(s) > 1`)

`[-4, -3, -2, 3, 2, 4]`  `6` `[-4, 3, -2, -3, 2, -4]`  `1`

Create a dictionary mapping each digit `d` to the lists of elements in `s` that end with `d`.

Using Built-In Functions & Comprehensions

What are the indices of all elements in a list `s` that have the smallest absolute value?

`[-4, -3, -2, 3, 2, 4]`
0 1 2 3 4 5  `[2, 4]` `[1, 2, 3, 4, 5]`  `[0]`

What's the largest sum of two adjacent elements in a list `s`? (Assume `len(s) > 1`)



`[-4, -3, -2, 3, 2, 4]`  `6` `[-4, 3, -2, -3, 2, -4]`  `1`

Create a dictionary mapping each digit `d` to the lists of elements in `s` that end with `d`.


`[5, 8, 13, 21, 34, 55, 89]`

Using Built-In Functions & Comprehensions


What are the indices of all elements in a list `s` that have the smallest absolute value?

`[-4, -3, -2, 3, 2, 4]`
`0 1 2 3 4 5`  `[2, 4]` `[1, 2, 3, 4, 5]`  `[0]`

What's the largest sum of two adjacent elements in a list `s`? (Assume `len(s) > 1`)



`[-4, -3, -2, 3, 2, 4]`  `6` `[-4, 3, -2, -3, 2, -4]`  `1`

Create a dictionary mapping each digit `d` to the lists of elements in `s` that end with `d`.

`[5, 8, 13, 21, 34, 55, 89]`  `{1: [21], 3: [13], 4: [34], 5: [5, 55], 8: [8], 9: [89]}`

Using Built-In Functions & Comprehensions


What are the indices of all elements in a list `s` that have the smallest absolute value?

`[-4, -3, -2, 3, 2, 4]`
`0 1 2 3 4 5`  `[2, 4]` `[1, 2, 3, 4, 5]`  `[0]`

What's the largest sum of two adjacent elements in a list `s`? (Assume `len(s) > 1`)

`[-4, -3, -2, 3, 2, 4]`  `6` `[-4, 3, -2, -3, 2, -4]`  `1`



Create a dictionary mapping each digit `d` to the lists of elements in `s` that end with `d`.

`[5, 8, 13, 21, 34, 55, 89]`  `{1: [21], 3: [13], 4: [34], 5: [5, 55], 8: [8], 9: [89]}`

Does every element equal some other element in `s`?

Using Built-In Functions & Comprehensions


What are the indices of all elements in a list `s` that have the smallest absolute value?

`[-4, -3, -2, 3, 2, 4]`
0 1 2 3 4 5  `[2, 4]` `[1, 2, 3, 4, 5]`  `[0]`


What's the largest sum of two adjacent elements in a list `s`? (Assume `len(s) > 1`)

`[-4, -3, -2, 3, 2, 4]`  `6` `[-4, 3, -2, -3, 2, -4]`  `1`

Create a dictionary mapping each digit `d` to the lists of elements in `s` that end with `d`.



`[5, 8, 13, 21, 34, 55, 89]`  `{1: [21], 3: [13], 4: [34], 5: [5, 55], 8: [8], 9: [89]}`

Does every element equal some other element in `s`?

`[-4, -3, -2, 3, 2, 4]`  `False`

Using Built-In Functions & Comprehensions


What are the indices of all elements in a list `s` that have the smallest absolute value?

`[-4, -3, -2, 3, 2, 4]`
0 1 2 3 4 5  `[2, 4]` `[1, 2, 3, 4, 5]`  `[0]`



What's the largest sum of two adjacent elements in a list `s`? (Assume `len(s) > 1`)

`[-4, -3, -2, 3, 2, 4]`  `6` `[-4, 3, -2, -3, 2, -4]`  `1`

Create a dictionary mapping each digit `d` to the lists of elements in `s` that end with `d`.

`[5, 8, 13, 21, 34, 55, 89]`  `{1: [21], 3: [13], 4: [34], 5: [5, 55], 8: [8], 9: [89]}`

Does every element equal some other element in `s`?

`[-4, -3, -2, 3, 2, 4]`  `False` `[4, 3, 2, 3, 2, 4]`  `True`

Examples: Linked Lists

Linked List Exercises

Linked List Exercises

Is a linked list sorted from least to greatest?

Linked List Exercises

Is a linked list `s` ordered from least to greatest?



Linked List Exercises

Is a linked list `s` ordered from least to greatest?



Linked List Exercises

Is a linked list `s` ordered from least to greatest?



Is a linked list `s` ordered from least to greatest by absolute value (or a key function)?

Linked List Exercises

Is a linked list *s* ordered from least to greatest?



Is a linked list *s* ordered from least to greatest by absolute value (or a key function)?



Linked List Exercises

Is a linked list `s` ordered from least to greatest?



Is a linked list `s` ordered from least to greatest by absolute value (or a key function)?



Linked List Exercises

Is a linked list *s* ordered from least to greatest?



Is a linked list *s* ordered from least to greatest by absolute value (or a key function)?



Create a sorted Link containing all the elements of both sorted Links *s* & *t*.

Linked List Exercises

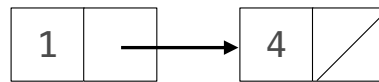
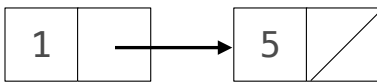
Is a linked list *s* ordered from least to greatest?



Is a linked list *s* ordered from least to greatest by absolute value (or a key function)?



Create a sorted Link containing all the elements of both sorted Links *s* & *t*.



Linked List Exercises

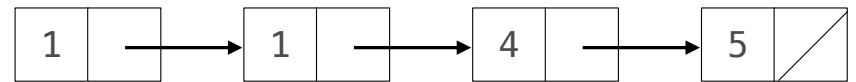
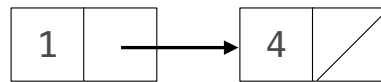
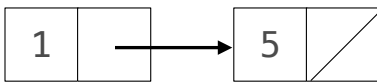
Is a linked list *s* ordered from least to greatest?



Is a linked list *s* ordered from least to greatest by absolute value (or a key function)?



Create a sorted Link containing all the elements of both sorted Links *s* & *t*.



Linked List Exercises

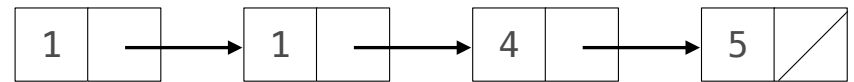
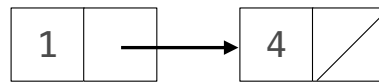
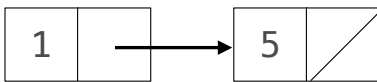
Is a linked list *s* ordered from least to greatest?



Is a linked list *s* ordered from least to greatest by absolute value (or a key function)?



Create a sorted Link containing all the elements of both sorted Links *s* & *t*.



Do the same thing, but never call Link.

Linked List Exercises

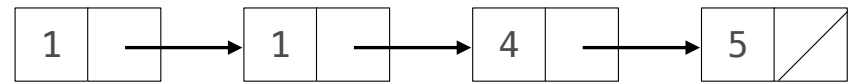
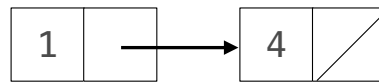
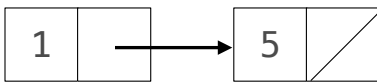
Is a linked list *s* ordered from least to greatest?



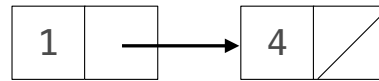
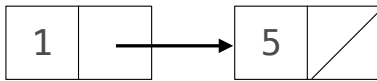
Is a linked list *s* ordered from least to greatest by absolute value (or a key function)?



Create a sorted Link containing all the elements of both sorted Links *s* & *t*.



Do the same thing, but never call Link.



Linked List Exercises

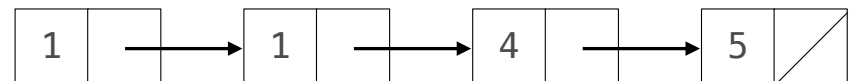
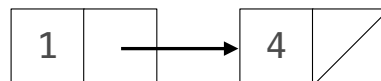
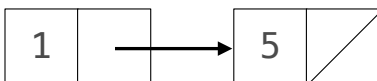
Is a linked list *s* ordered from least to greatest?



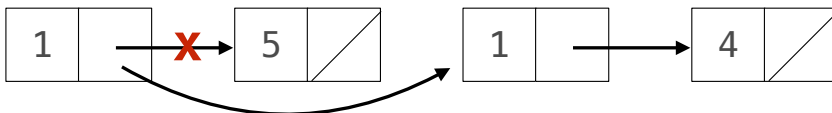
Is a linked list *s* ordered from least to greatest by absolute value (or a key function)?



Create a sorted Link containing all the elements of both sorted Links *s* & *t*.



Do the same thing, but never call Link.

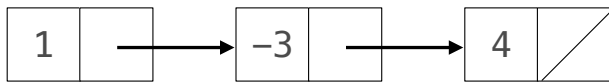


Linked List Exercises

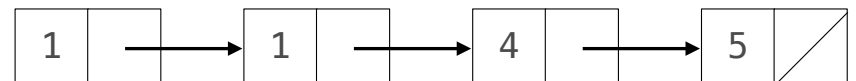
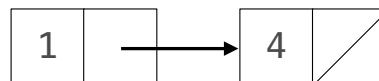
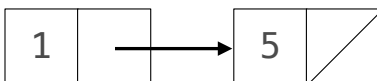
Is a linked list *s* ordered from least to greatest?



Is a linked list *s* ordered from least to greatest by absolute value (or a key function)?



Create a sorted Link containing all the elements of both sorted Links *s* & *t*.



Do the same thing, but never call Link.

