New today! Go to **Course.Care** > **COMP110** > **Join Discussion**
If you want to ask a question in lecture from 11am EST to 12:15am EST!

Fall 2021

Class 24 - Dictionary Practice

Announcements

- Kaki's Review Session this afternoon at 4pm in SN014
- Optional Practice for Quiz - Posted on Course Site under Quiz 4 and Resources
- Deliverables:
  - EX08 - Due tonight at 11:59pm
  - RD02 - Due LDOC at 11:59pm
  - Final exercise of semester will go out later this week and be due LDOC at 11:59pm
  - No late hand-ins after LDOC
- Quiz 4 - Thursday with a focus on Object-oriented Programming
  - Classes vs Objects
  - Constructors
  - Methods and Method Calls
  - self
  - Operator Overloads
  - Union and isinstance

```python
class Rat:
    n: int
    d: int

    def __init__(self, n: int, d: int):
        self.n = n
        self.d = d

    def __repr__(self) -> str:
        return f"{self.n}/{self.d}"

    def simplify(self) -> Rat:
        gcd = self.d // 2
        while gcd > 1:
            if self.n % gcd == 0 and self.d % gcd == 0:
                return Rat(self.n // gcd, self.d // gcd)
            else:
                gcd -= 1
        return self


x = Rat(6, 8)
y = Rat(1, 2)
```

Part 2 will continue here...

Diagram 1 - Part 1

Produce an environment diagram of the code listing left.

```python
class Rat:
    n: int
    d: int

    def __init__(self, n: int, d: int):
        self.n = n
        self.d = d

    def __repr__(self) -> str:
        return f"{self.n}/{self.d}"

    def simplify(self) -> Rat:
        gcd = self.d // 2
        while gcd > 1:
            if self.n % gcd == 0 and self.d % gcd == 0:
                return Rat(self.n // gcd, self.d // gcd)
            else:
                gcd -= 1
        return self


x = Rat(6, 8)
y = Rat(1, 2)

simp_y = y.simplify()
print(y)
```

Part 2 will continue here...

```python
class Rat:
    n: int
    d: int

    def __init__(self, n: int, d: int):
        self.n = n
        self.d = d

    def __repr__(self) -> str:
        return f"{self.n}/{self.d}"

    def simplify(self) -> Rat:
        gcd = self.d // 2
        while gcd > 1:
            if self.n % gcd == 0 and self.d % gcd == 0:
                return Rat(self.n // gcd, self.d // gcd)
            else:
                gcd -= 1
        return self


x = Rat(6, 8)
y = Rat(1, 2)

simp_y = y.simplify()
print(simp_y)

simp_x = x.simplify()
print(simp_x)
```

Diagram 1 - Part 2

Continue the diagram with outlined statements left.

```python
class Rat:
    n: int
    d: int

    def __init__(self, n: int, d: int):
        self.n = n
        self.d = d

    def __repr__(self) -> str:
        return f"{self.n}/{self.d}"

    def simplify(self) -> Rat:
        gcd = self.d // 2
        while gcd > 1:
            if self.n % gcd == 0 and self.d % gcd == 0:
                return Rat(self.n // gcd, self.d // gcd)
            else:
                gcd -= 1
        return self


x = Rat(6, 8)
y = Rat(1, 2)

simp_y = y.simplify()
print(simp_y)

simp_x = x.simplify()
print(simp_x)
```

```
3   class Rat:
4       n: int = 1
5       d: int = 1
6
7       def __repr__(self) -> str:
8           return f"{self.n}/{self.d}"
9
10      def __add__(self, rhs: Rat) -> Rat:
11          r = Rat()
12          r.n = self.n * rhs.d + rhs.n * self.d
13          r.d = self.d * rhs.d
14          return r
15
16
17  x = Rat()
18  x.n = 1
19  x.d = 5
20  y = Rat()
21  y.n = 0
22  y.d = 4
23
24  z = y + x
25  print(z)
```

Diagram 2

Complete the diagram left.

```python
class Rat:
    n: int = 1
    d: int = 1

    def __repr__(self) -> str:
        return f"{self.n}/{self.d}"

    def __add__(self, rhs: Rat) -> Rat:
        r = Rat()
        r.n = self.n * rhs.d + rhs.n * self.d
        r.d = self.d * rhs.d
        return r


x = Rat()
x.n = 1
x.d = 5
y = Rat()
y.n = 0
y.d = 4

z = y + x
print(z)
```

```python
class Node:
    v: int
    next: Union[Node, None]

    def __init__(self, v: int, next: Union[Node, None]):
        self.v = v
        self.next = next

    def __repr__(self) -> str:
        r: str = f"{self.v} -> "
        n: Union[Node, None] = self.next
        while isinstance(n, Node):
            r += f"{n.v} -> "
            n = n.next
        r += "None"
        return r


n1 = Node(1, None)
n2 = Node(2, n1)
```

Part 2 will continue here...

Diagram 3 - Part 1

Complete the diagram left.

```python
class Node:
    v: int
    next: Union[Node, None]

    def __init__(self, v: int, next: Union[Node, None]):
        self.v = v
        self.next = next

    def __repr__(self) -> str:
        r: str = f"{self.v} -> "
        n: Union[Node, None] = self.next
        while isinstance(n, Node):
            r += f"{n.v} -> "
            n = n.next
        r += "None"
        return r


n1 = Node(1, None)
n2 = Node(2, n1)
```

Part 2 will continue here...

```python
class Node:
    v: int
    next: Union[Node, None]

    def __init__(self, v: int, next: Union[Node, None]):
        self.v = v
        self.next = next

    def __repr__(self) -> str:
        r: str = f"{self.v} -> "
        n: Union[Node, None] = self.next
        while isinstance(n, Node):
            r += f"{n.v} -> "
            n = n.next
        r += "None"
        return r


n1 = Node(1, None)
n2 = Node(2, n1)

print(n2)
```

Diagram 3 - Part 2

Complete the diagram left.

```python
class Node:
    v: int
    next: Union[Node, None]

    def __init__(self, v: int, next: Union[Node, None]):
        self.v = v
        self.next = next

    def __repr__(self) -> str:
        r: str = f"{self.v} -> "
        n: Union[Node, None] = self.next
        while isinstance(n, Node):
            r += f"{n.v} -> "
            n = n.next
        r += "None"
        return r


n1 = Node(1, None)
n2 = Node(2, n1)

print(n2)
```

```python
class Node:
    v: int
    next: Union[Node, None]

    def __init__(self, v: int, next: Union[Node, None]):
        self.v = v
        self.next = next

    def __repr__(self) -> str:
        s: str = f"{self.v} -> "
        if isinstance(self.next, Node):
            return s + self.next.__repr__()
        else:
            return s + "None"


n1 = Node(1, None)
n2 = Node(2, n1)
n3 = Node(3, n2)

print(n3)
```

Diagram 3 - Bonus Content

Complete the diagram left.