APS106

UNIVERSITY OF TORONTO

# binary search trees.

**Week 12** | Lecture 2 (12.2)

if nothing else, write #cleancode

# This Week's Content

- **Lecture 12.1**
  - Linked lists, binary trees
  - Reading: Chapter 14
- **Lecture 12.2**
  - **Binary search trees**
  - **Reading: Chapter 14**
- **Lecture 12.3**
  - Design Problem: 20 Questions

# Clearing things up.

```python
class LinkedList:

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

```python
class LinkedList:

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next is not None:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

```python
class LinkedList:

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next != None:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

```
class LinkedList:

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next is not None:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

# What are we testing?

- `is` is an identity test.
- It checks whether the right-hand side and the left-hand side are the very same object.

```
>>> a = 'hello world'
>>> b = 'hello world'
>>> a is b
False

>>> id(a)
1603648396784

>>> id(b)
1603648426160
```

```
class LinkedList:

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next is not None:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

# What are we testing?

- `is` is an identity test.
- It checks whether the right-hand side and the left-hand side are the very same object.

```
>>> a = None
>>> b = None
>>> a is b
True

>>> id(a)
140718929239264

>>> id(b)
140718929239264
```

```
class LinkedList:

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next is not None:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

# What are we testing?

- **is** is an identity test.
- It checks whether the right-hand side and the left-hand side are the very same object.

```
while on.next is not None:
```

```
>>> Node() is not None
True
```

```
>>> None is not None
False
```

```python
class LinkedList:

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next != None:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

# What are we testing?

- **==** is an equality test.
- It checks whether the right-hand side and the left-hand side are equal objects.

```
>>> a = 'hello world'
>>> b = 'hello world'
>>> a == b
True

>>> id(a)
1603648396784

>>> id(b)
1603648426160
```

```
class LinkedList:

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next != None:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

# What are we testing?

- == is an equality test.
- It checks whether the right-hand side and the left-hand side are equal objects.

```
while on.next != None:
```

```
>>> Node() != None
True

>>> None != None
False
```

```
class LinkedList:

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

# What are we testing?

- Truthy and Falsy Values in Python.
- Expressions with operands and operators evaluate to either `True` or `False` and they can be used in an `if` or `while` condition to determine if a code block should run.

```
>>> if 5 > 3:
        print("True")
True
```

```
class LinkedList:

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

# What are we testing?

- Truthy and Falsy Values in Python.
- What do you think would be the output of this code?

```
>>> a = 4
>>> if a:
        print(a)
```

```
class LinkedList:

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

# What are we testing?

- Truthy and Falsy Values in Python.
- What do you think would be the output of this code?

```
>>> a = 4
>>> if a:
        print(a)
4
```

```
class LinkedList:

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

# What are we testing?

- Truthy and Falsy Values in Python.
- What do you think would be the output of this code?

```
>>> a = 0
>>> if a:
        print(a)
```

```
class LinkedList:

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

# What are we testing?

- Truthy and Falsy Values in Python.
- In Python, individual values can evaluate to either True or False.
- They do not necessarily have to be part of a larger expression to evaluate to a truth value because they already have one that has been determined by the rules of the Python language?
  - Values that evaluate to `False` are considered **Falsy**.
  - Values that evaluate to `True` are considered **Truthy**.

```
class LinkedList:

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next:  ←
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

# What are we testing?

- **Falsy Values**
- **Sequences and Collections**
    - Empty lists `[]`
    - Empty tuples `()`
    - Empty dictionaries `{}`
    - Empty sets `set()`
    - Empty strings `""`
    - Empty ranges `range(0)`
- **Numbers**
    - Zero of any numeric type.
    - Integer: `0`
    - Float: `0.0`
    - Complex: `0j`
- **Constants**
    - `None` ←
    - `False`

```
class LinkedList:

    def __init__(self):
        """
        (self) -> NoneType
        Create an empty linked list.
        """
        self.length = 0
        self.head = None

    def __str__(self): ...

    def add_to_head(self, cargo): ...

    def add_to_tail(self, cargo):
        """
        (self, object) -> NoneType
        Add cargo to the tail of the list.
        """
        on = self.head

        while on.next:
            on = on.next

        on.next = Node(cargo)

    def get_at_index(self, index): ...

    def delete_by_cargo(self, cargo): ...
```

# What are we testing?

- **Truthy Values**
- By default, an object is considered `True`.
- **Non-empty** sequences or collections (`lists`, `tuples`, `strings`, `dictionaries`, `sets`).
- **Numeric values** that are **not zero**.
- `True`

# Clearing things up

- When in **doubt**, try it **out**!

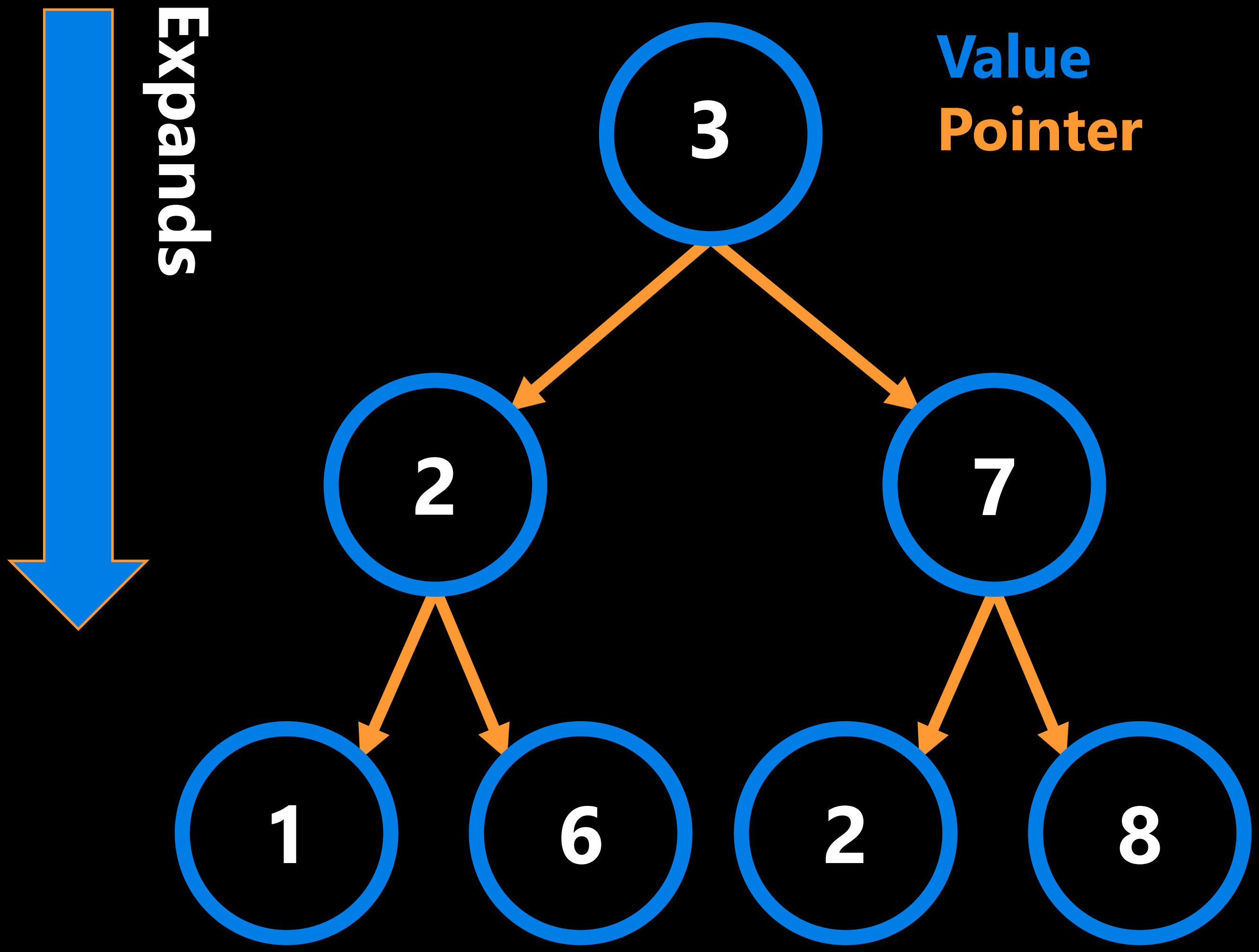
**Open your notebook**

**Click Link:**

**1. Truthy and Falsy Values**

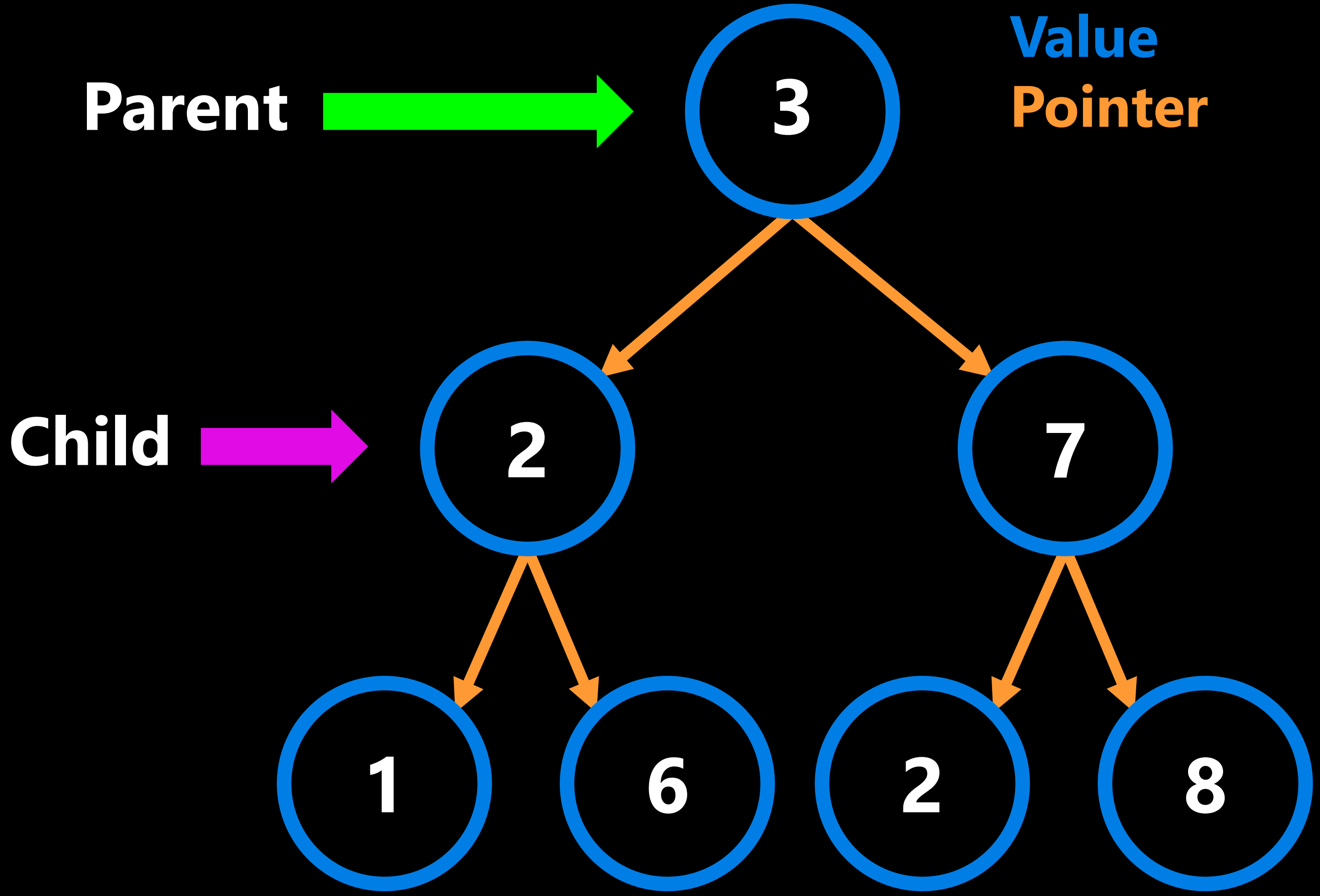
# Trees

- **Trees expand in one direction.**
- Trees are made up of parents and children.
  - These are relative terms for nodes.
  - Every parent can be a child.



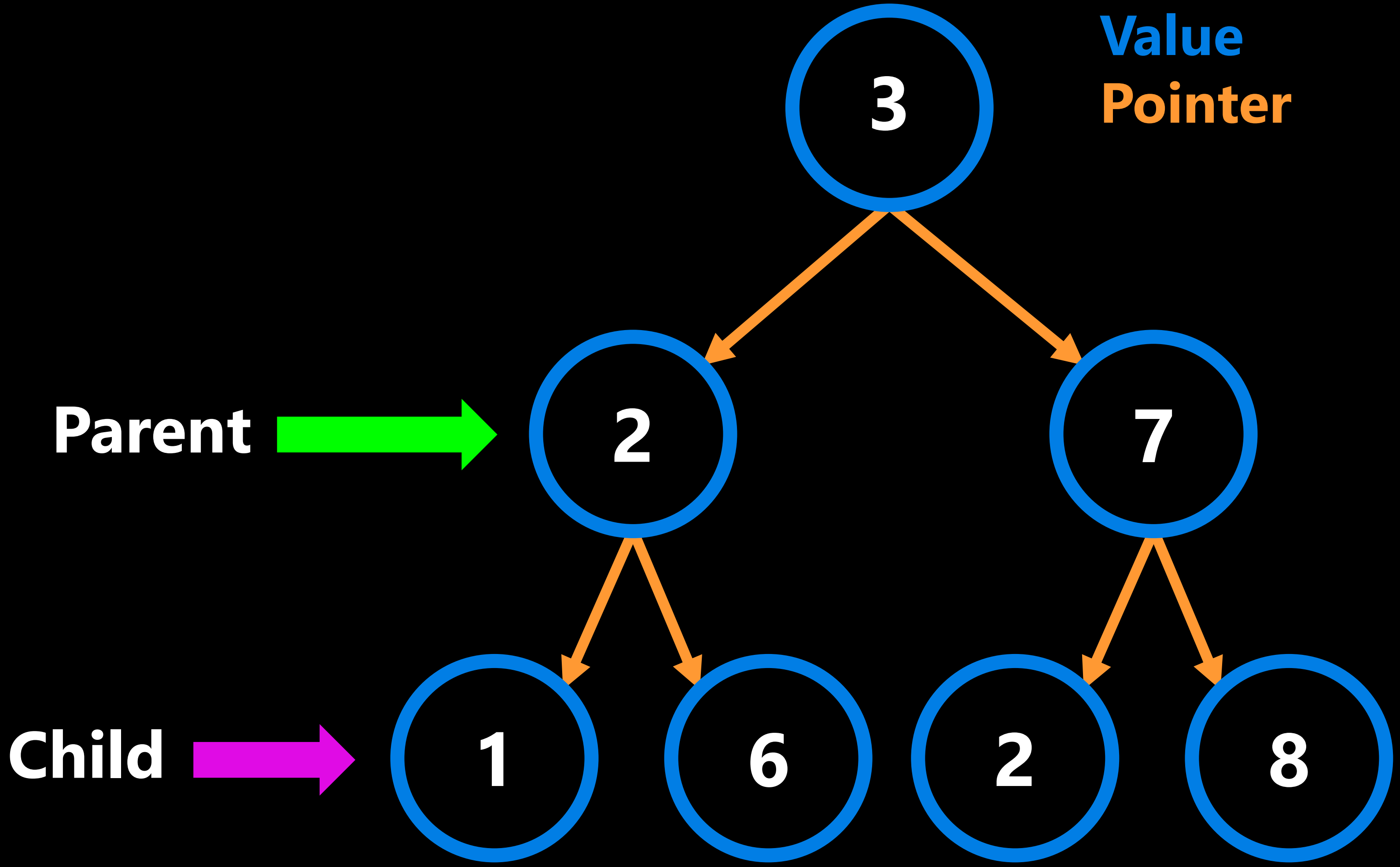
# Trees

- Trees expand in one direction.
- **Trees are made up of parents and children.**
  - **These are relative terms for nodes.**
  - **Every parent can be a child.**



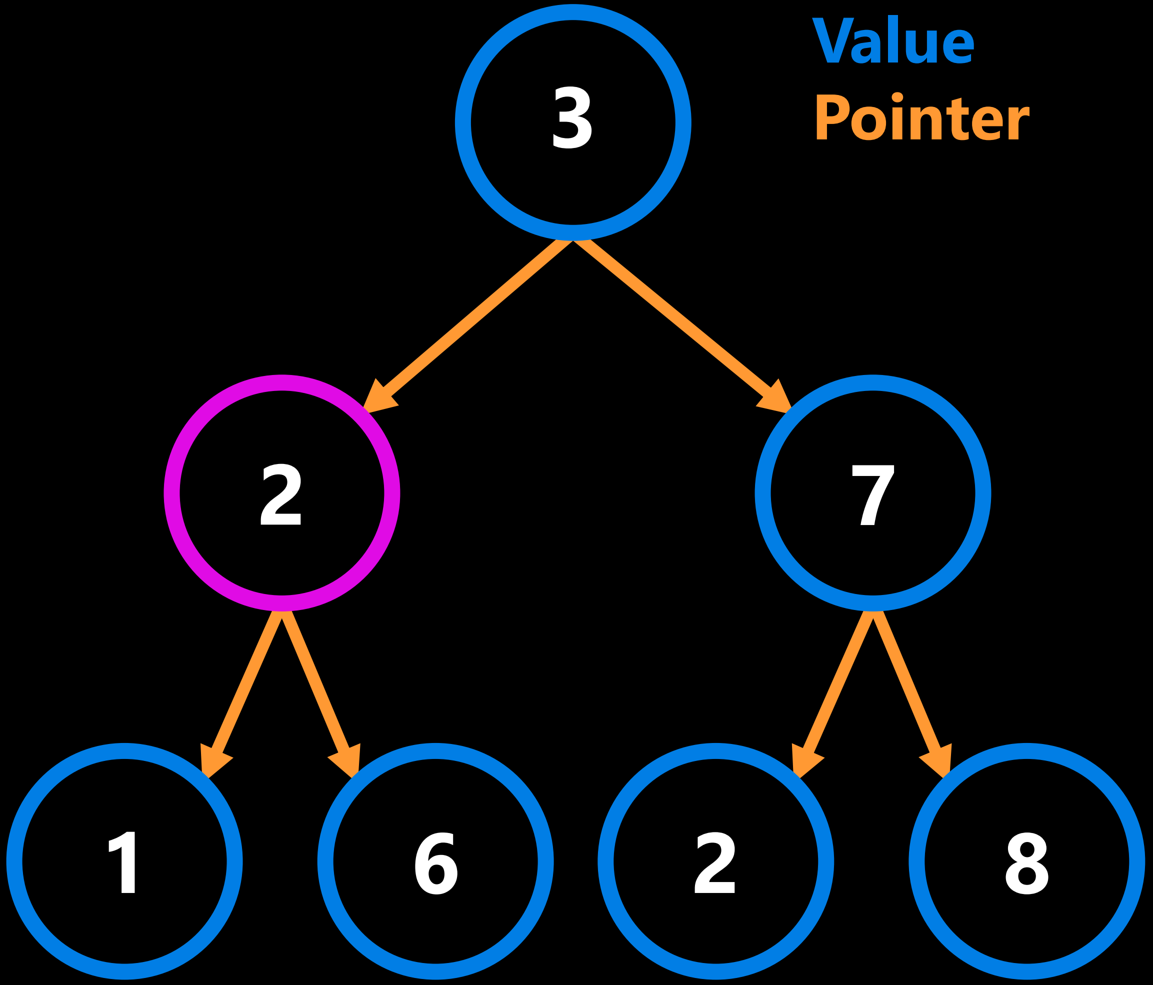
# Trees

- Trees expand in one direction.
- **Trees are made up of parents and children.**
  - **These are relative terms for nodes.**
  - **Every parent can be a child.**



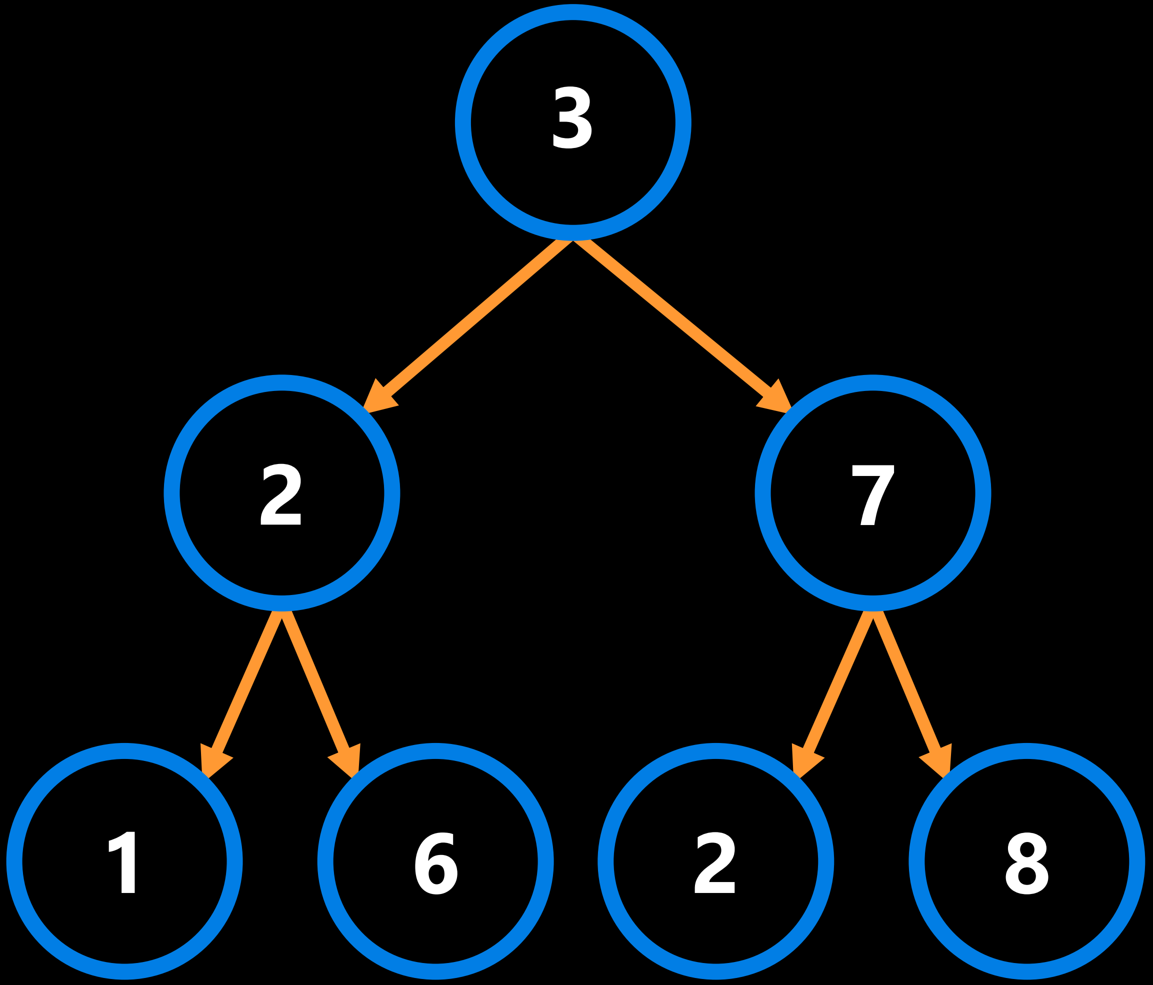
# Trees

- Node 2 is a child of Node 3 and a parent of Node 1 and Node 6.
- Every node can only have one parent but can have many children.



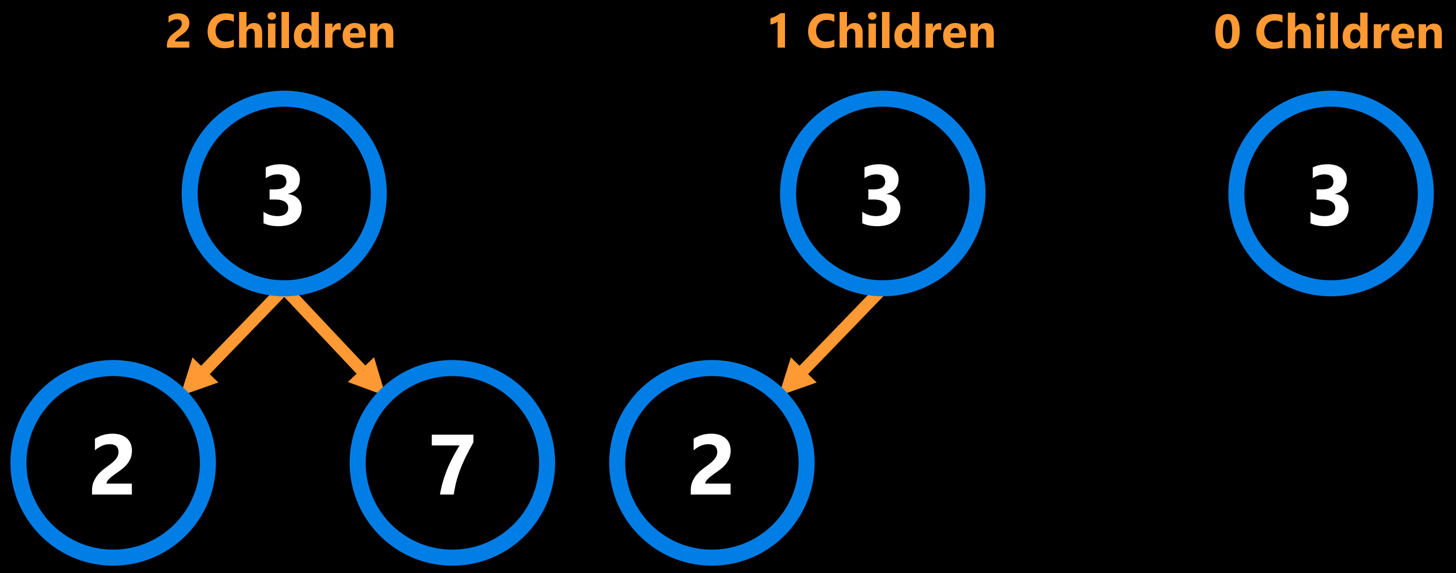
# Trees

- There are many different types of trees.
  - Family Trees.
  - Decision Trees.
  - Heaps.
  - Tries.
  - HTML Trees.
  - **Binary Trees (We will focus on these).**



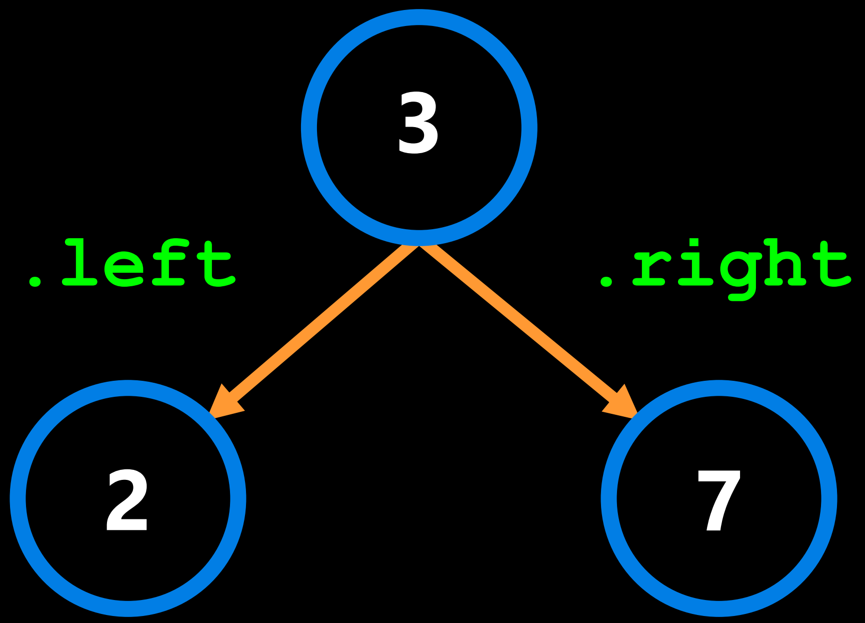
# Binary Trees

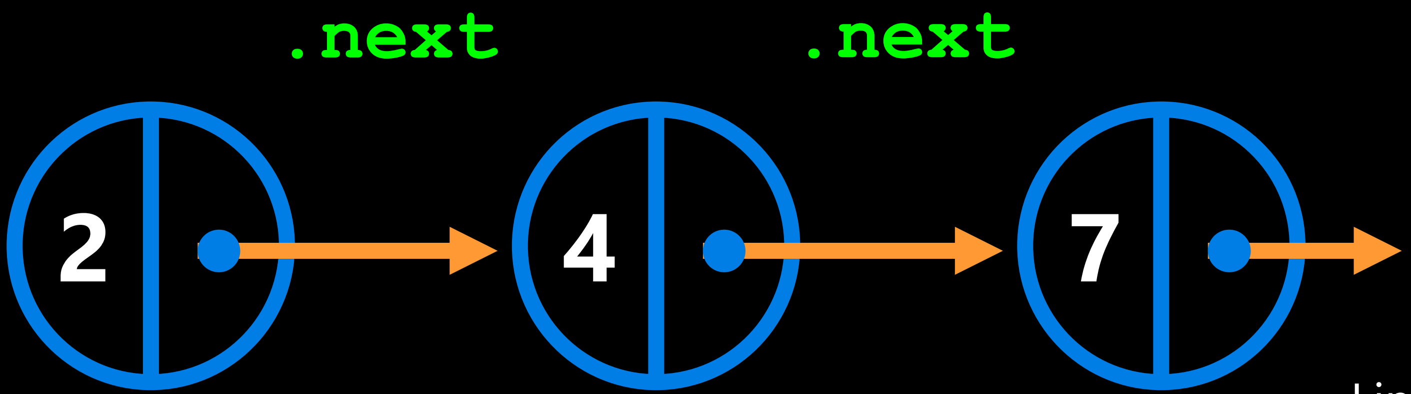
- Main Rule:
  - Each Node can have a maximum of two children (Pointers).
    - **0 Children**
    - **1 Children**
    - **2 Children**



# Binary Trees

- Children are represented using `.left` and `.right`.

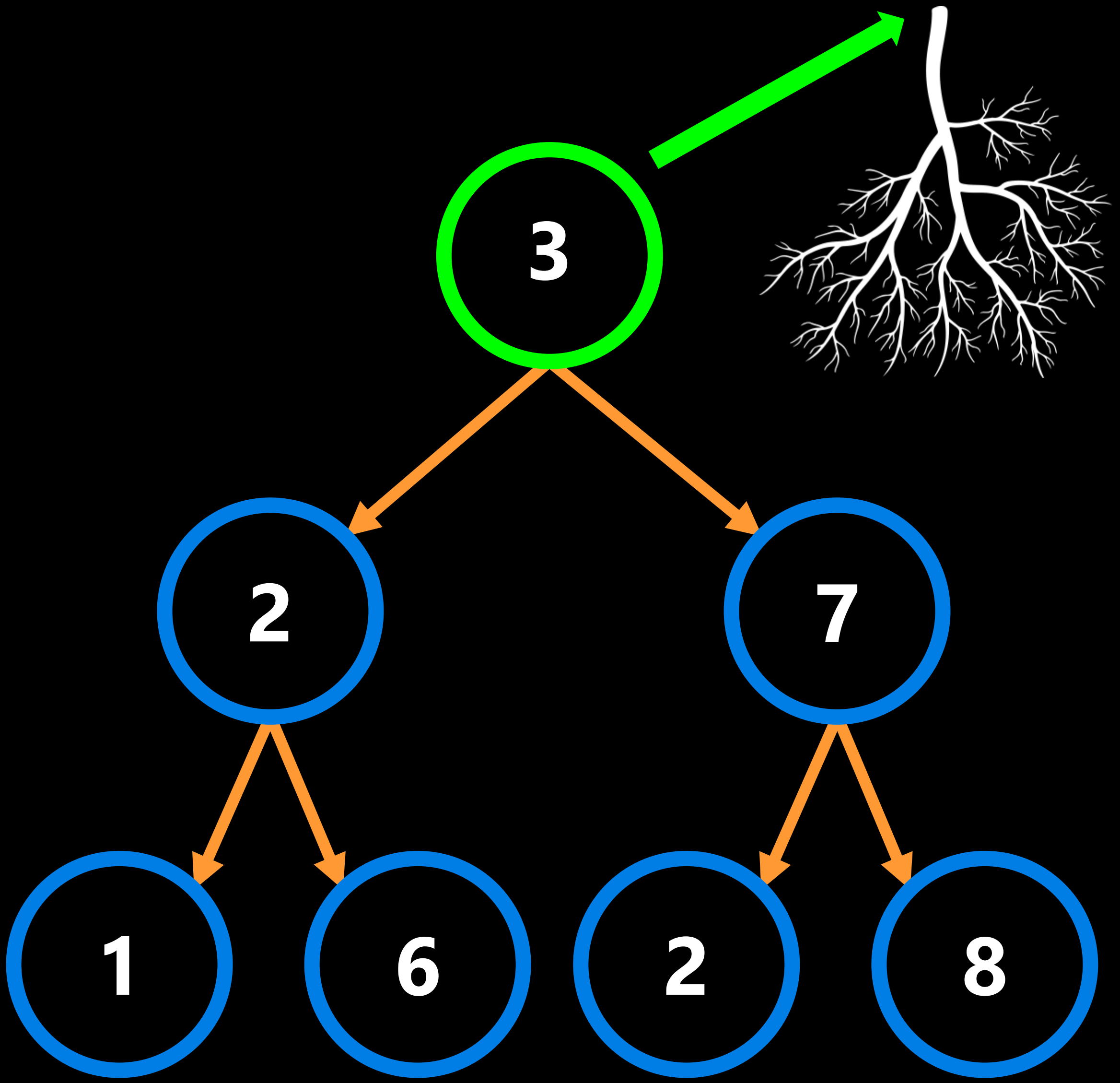




Linked list for reference.

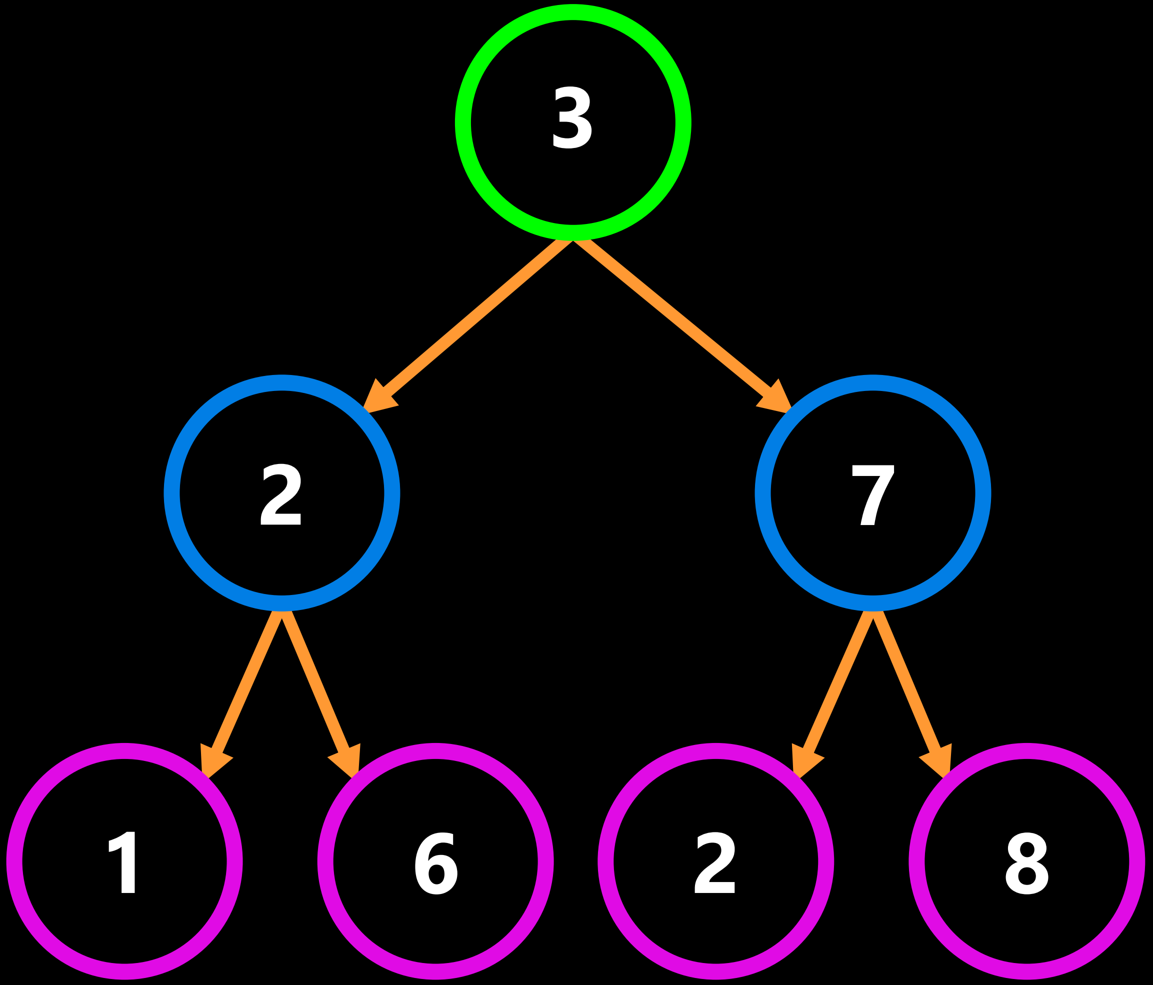
# Binary Trees

- **Terminology**
- The top node is called the **root node**.
- Any node without children is called a leaf node.
- The path between the root node and a leaf node is called a branch.



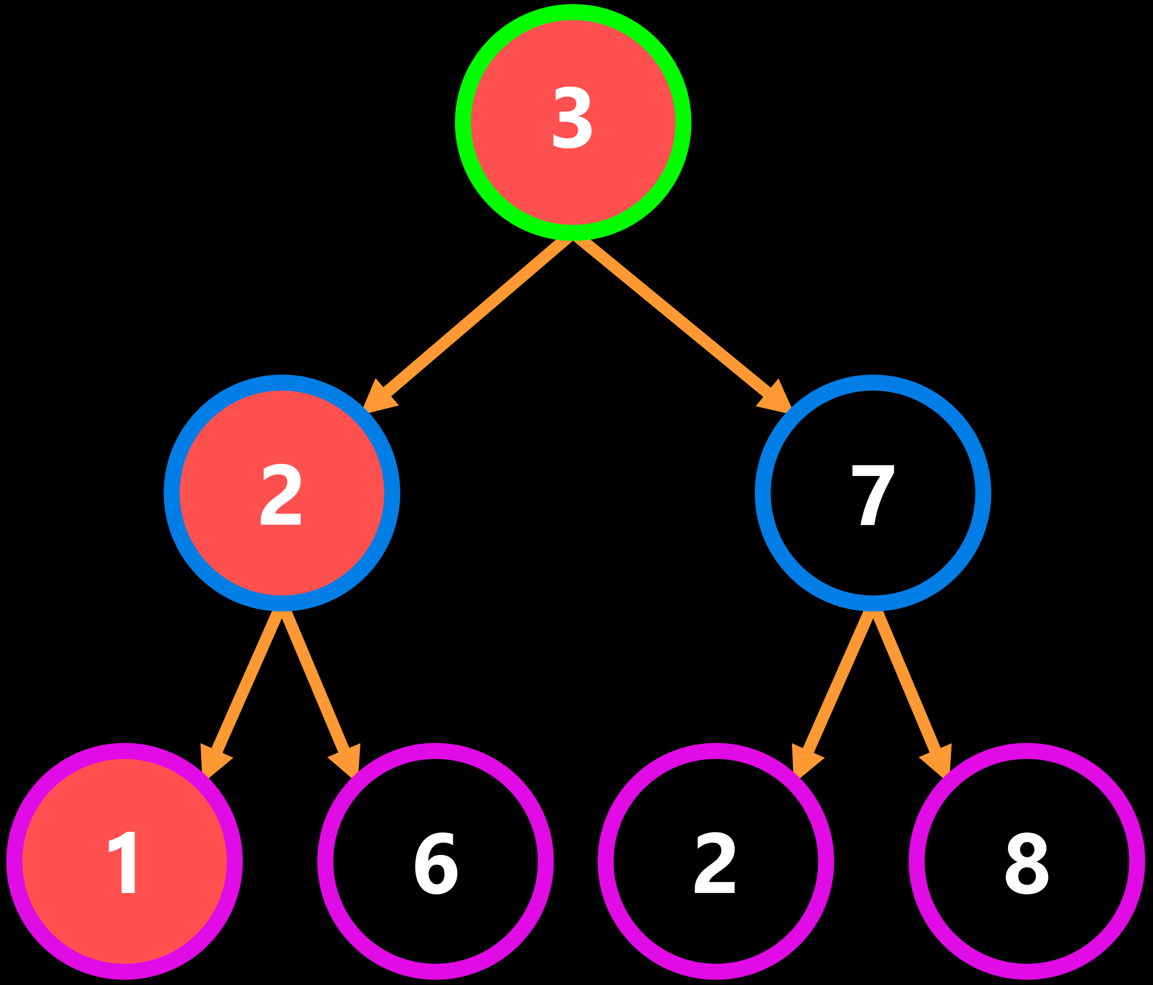
# Binary Trees

- **Terminology**
- The top node is called the **root node**.
- Any node without children is called a **leaf node**.
- The path between the root node and a leaf node is called a branch.



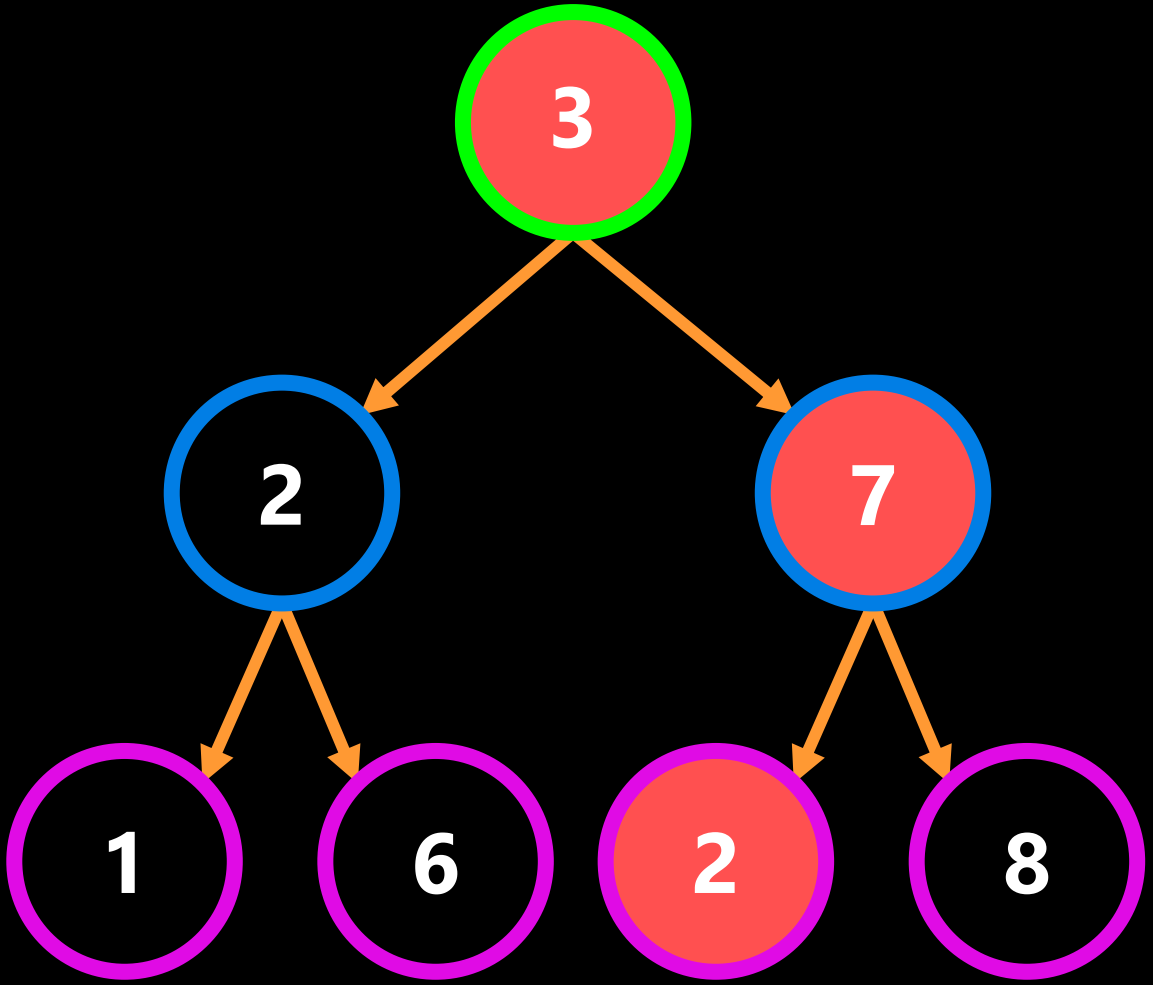
# Binary Trees

- **Terminology**
- The top node is called the **root node**.
- Any node without children is called a **leaf node**.
- The path between the root node and a leaf node is called a **branch**.



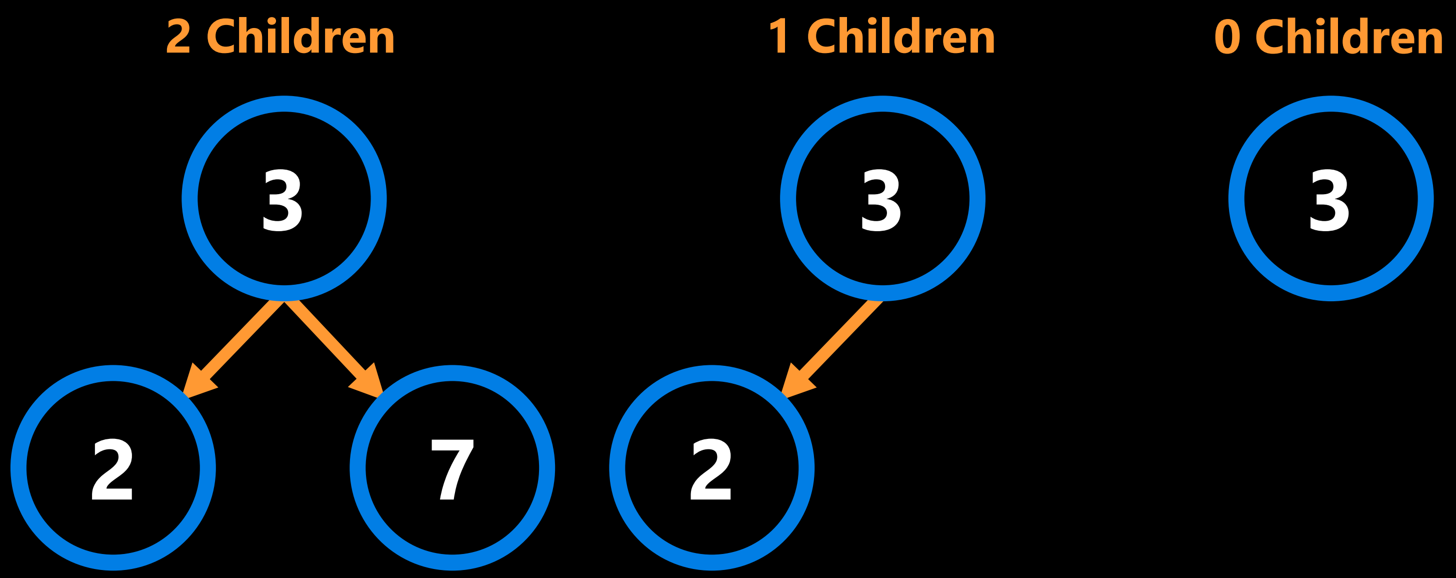
# Binary Trees

- **Terminology**
- The top node is called the **root node**.
- Any node without children is called a **leaf node**.
- The path between the root node and a leaf node is called a **branch**.



# Binary Trees

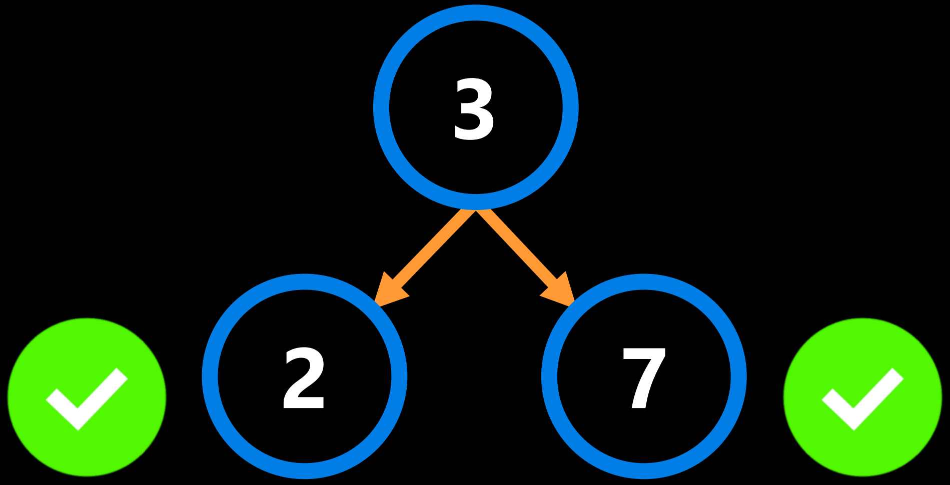
- Main Rule:
  - Each Node can have a maximum of two children (Pointers).
    - **0 Children**
    - **1 Children**
    - **2 Children**



# Binary Search Trees

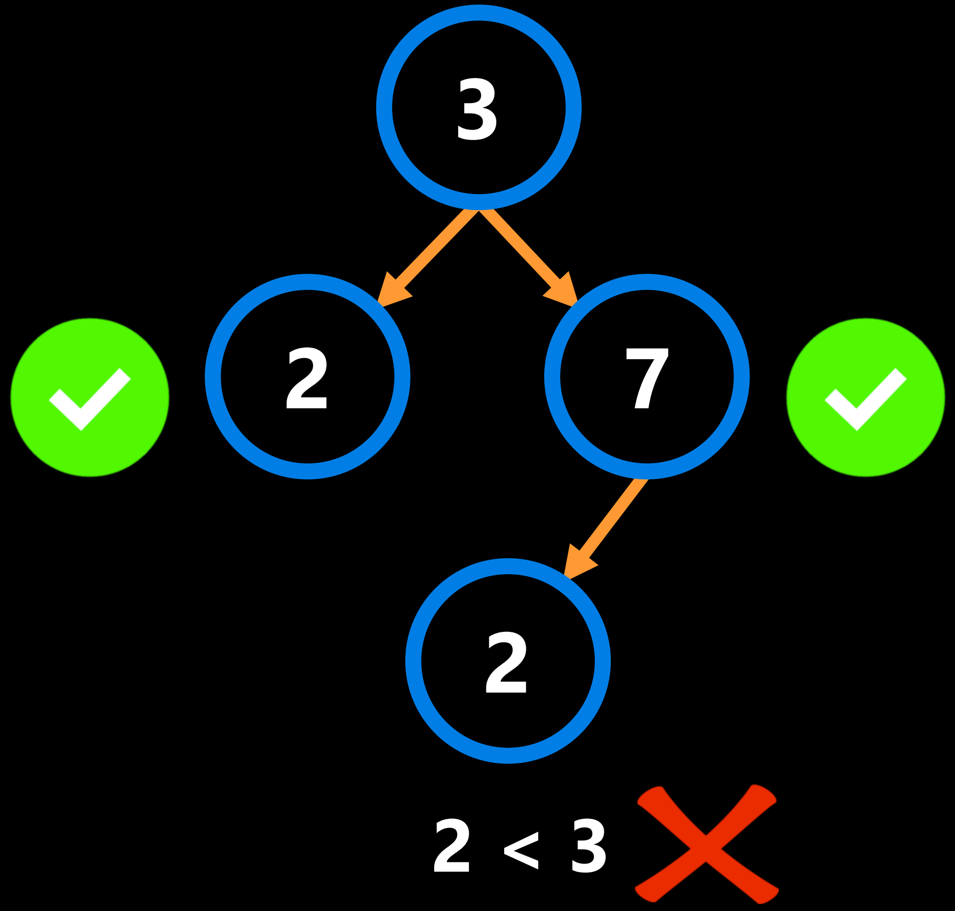
A special case of the binary tree.

- Main Rule:
  - Each Node can have a maximum of two children (Pointers).
    - 0 Children
    - 1 Children
    - 2 Children
  - `node.cargo` must be more than `node.left.cargo` and less than `node.right.cargo`.
    - 3 > 2
    - 3 < 7



# Binary Search Trees

- Main Rule:
  - Each Node can have a maximum of two children (Pointers).
    - 0 Children
    - 1 Children
    - 2 Children
  - `node.cargo` must be more than `node.left.cargo` and less than `node.right.cargo`.
    - 3 > 2
    - 3 < 7
  - This rule must be true for the entire tree.
    - Everything to the right of 3 must be greater than 3.



# The Binary Search Tree Class

- Let’s check out the BinarySearchTree class functionality.

## Open your notebook

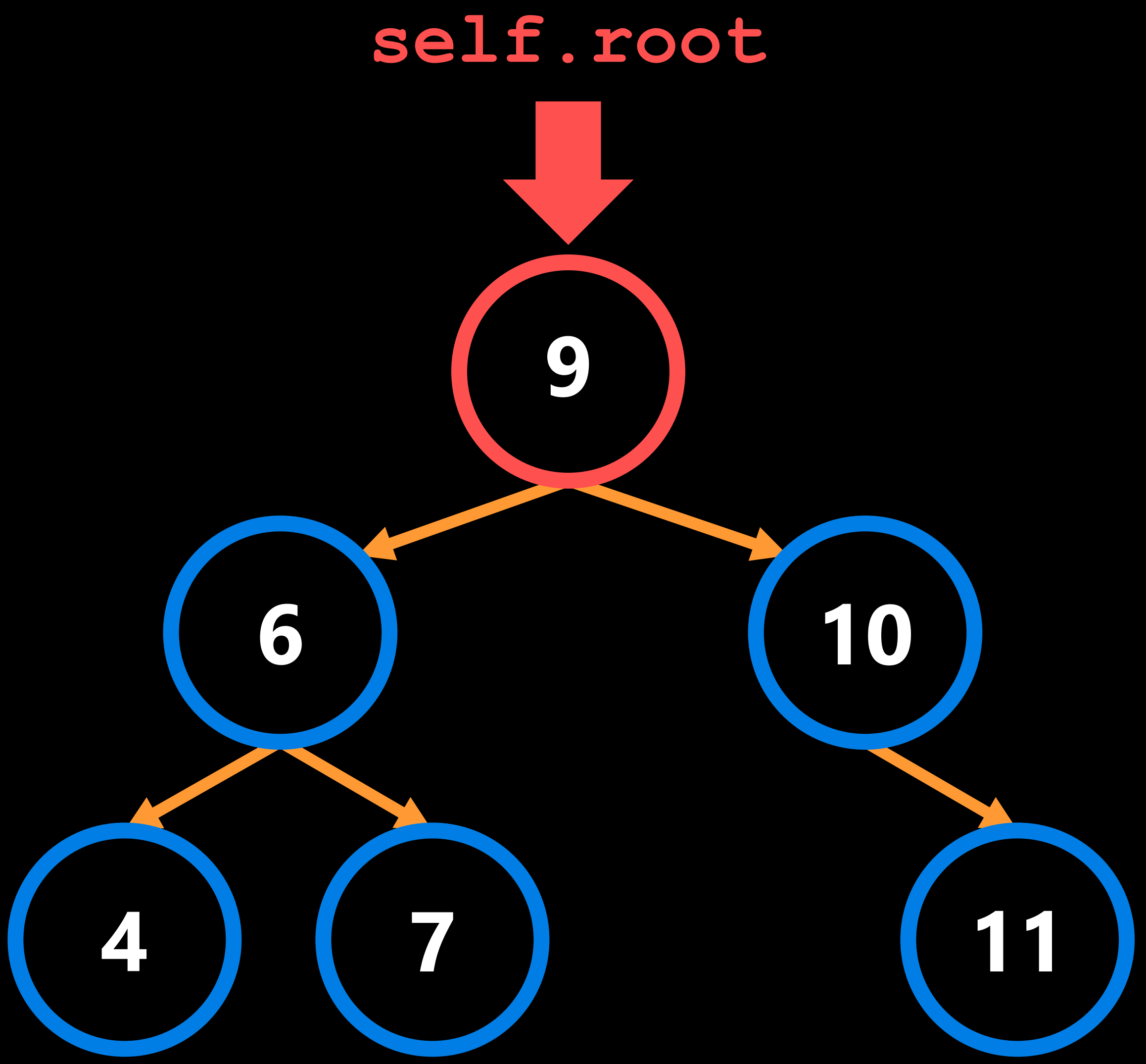
**Click Link:**

**2. BinarySearchTree Class**

# Let's try with a valid binary search tree.

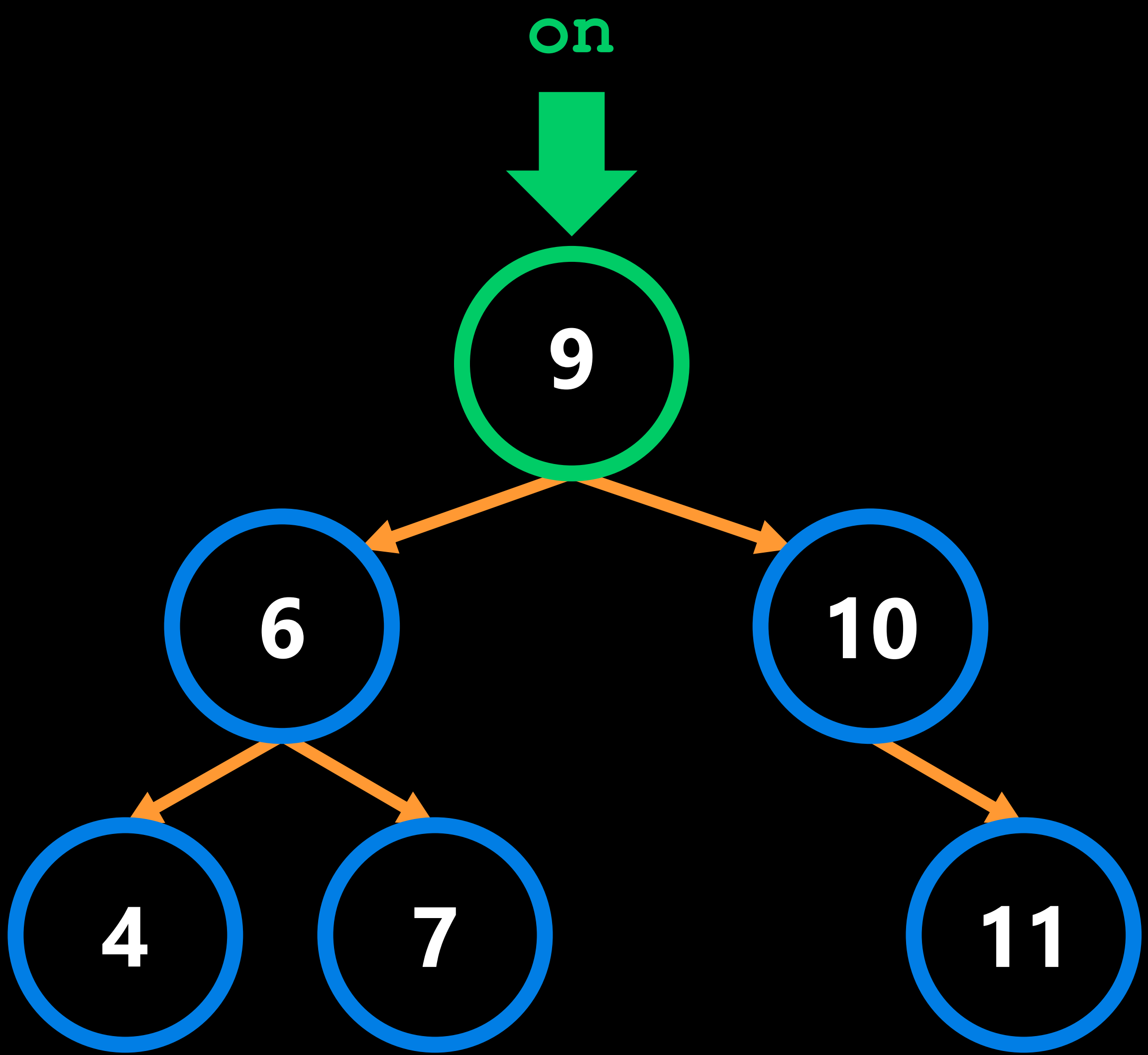
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

# This is a Valid Tree



```python
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

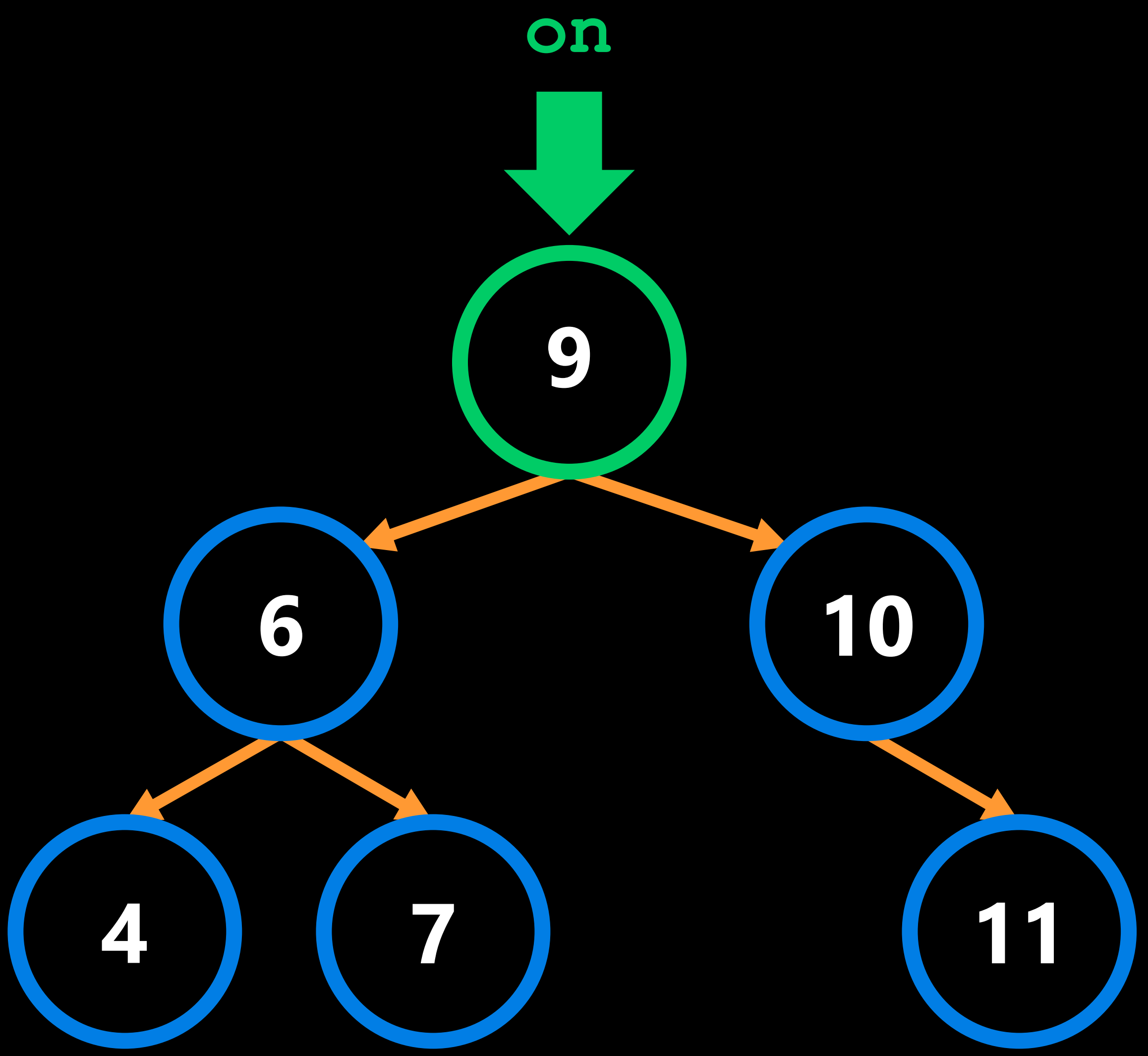
**Set on position.**



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

Create stack list.

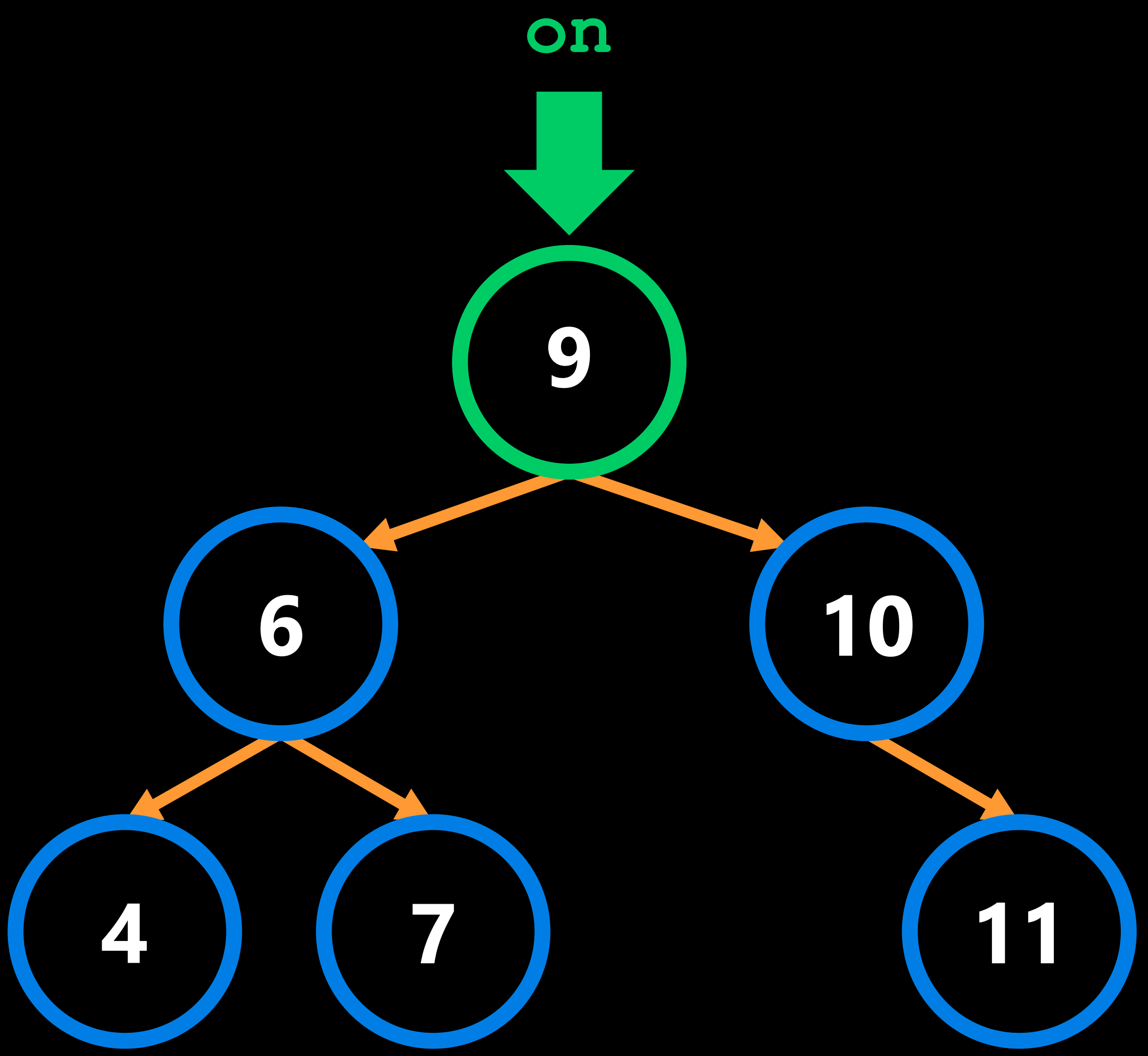
stack = [ ]



```python
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

**Initialize previous node.**

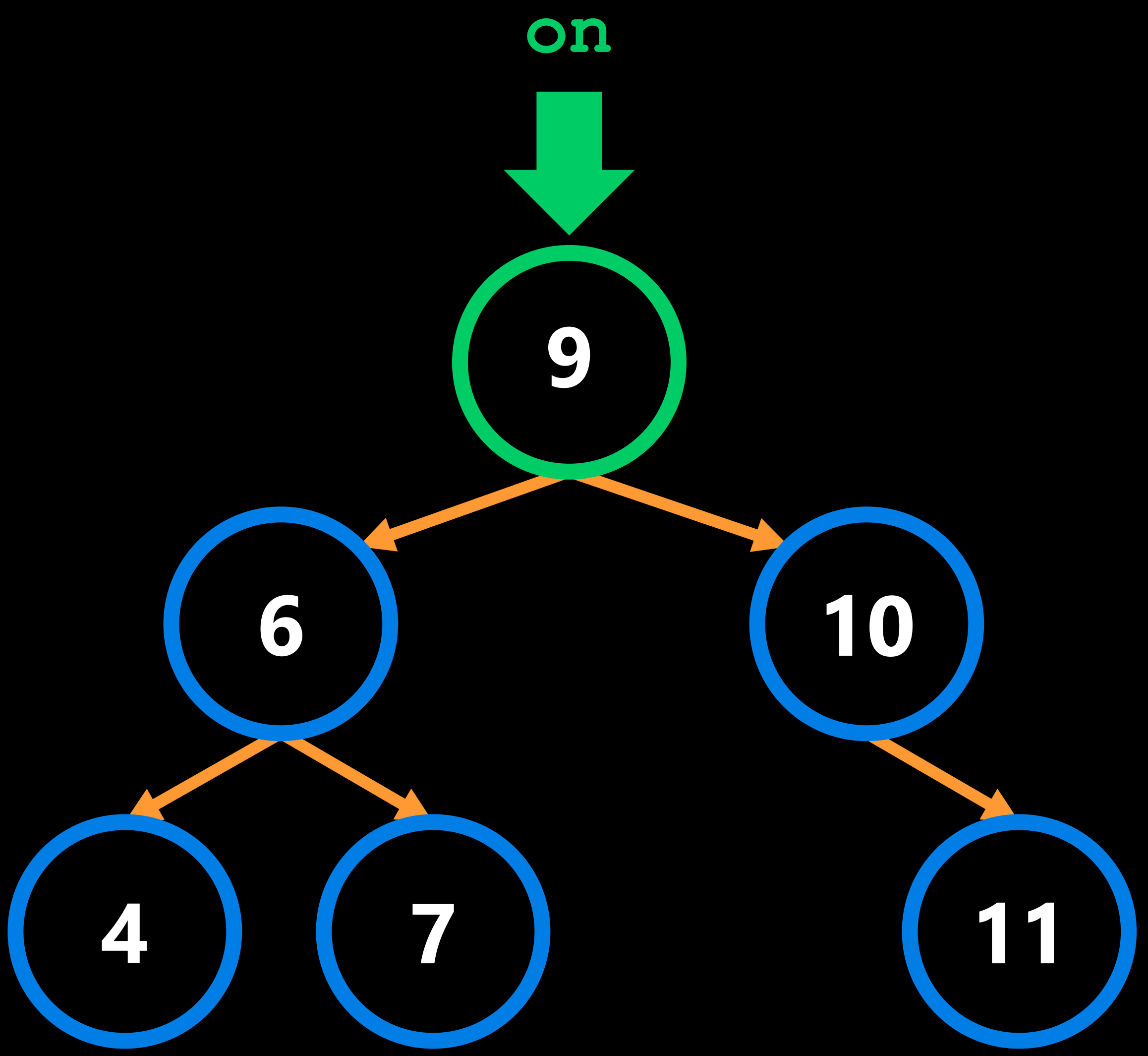
**stack = [ ]**



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

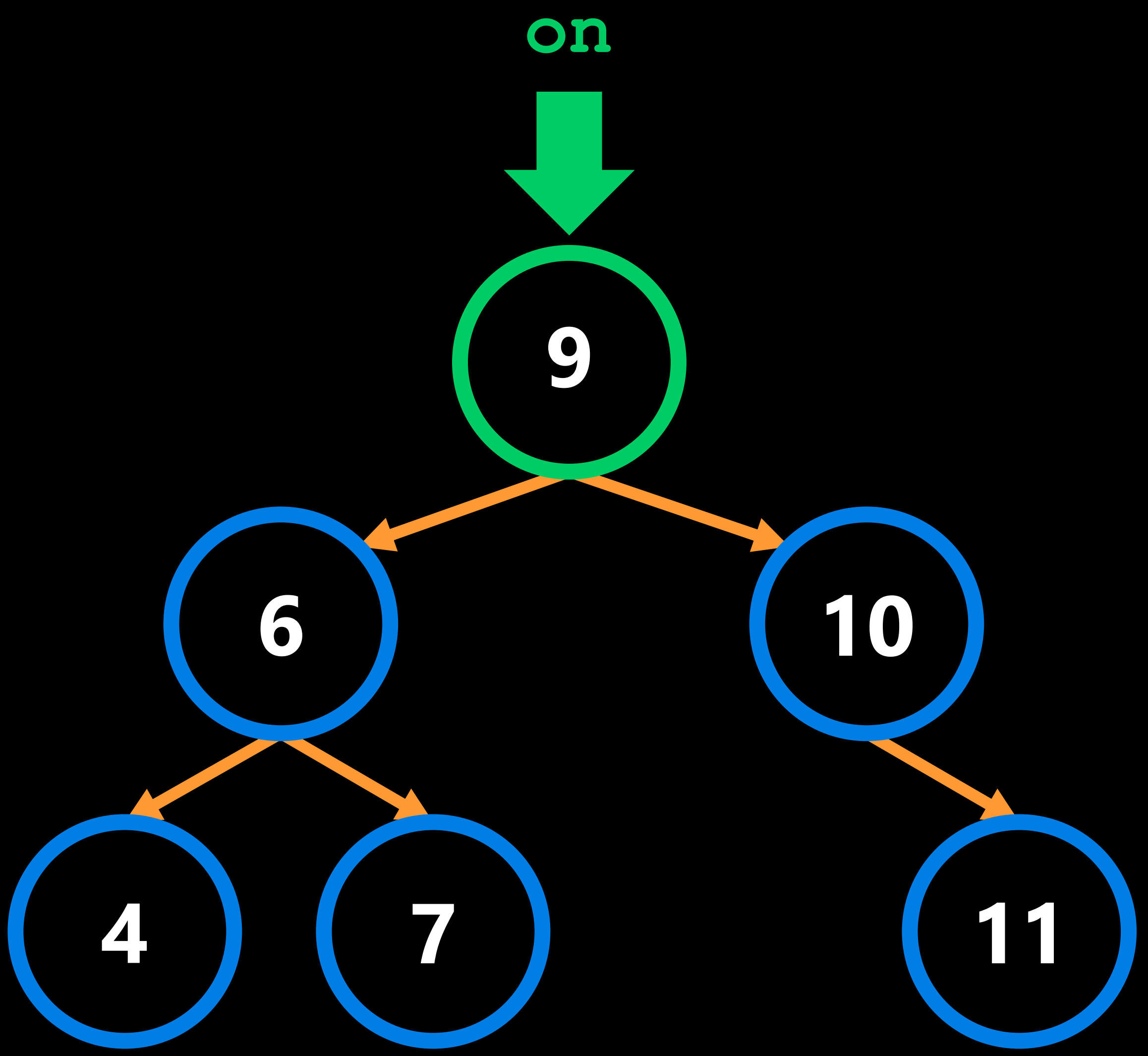
True

stack = [ ]



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:  ⬅ True

            while on is not None:  ⬅ True
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

stack = [ ]



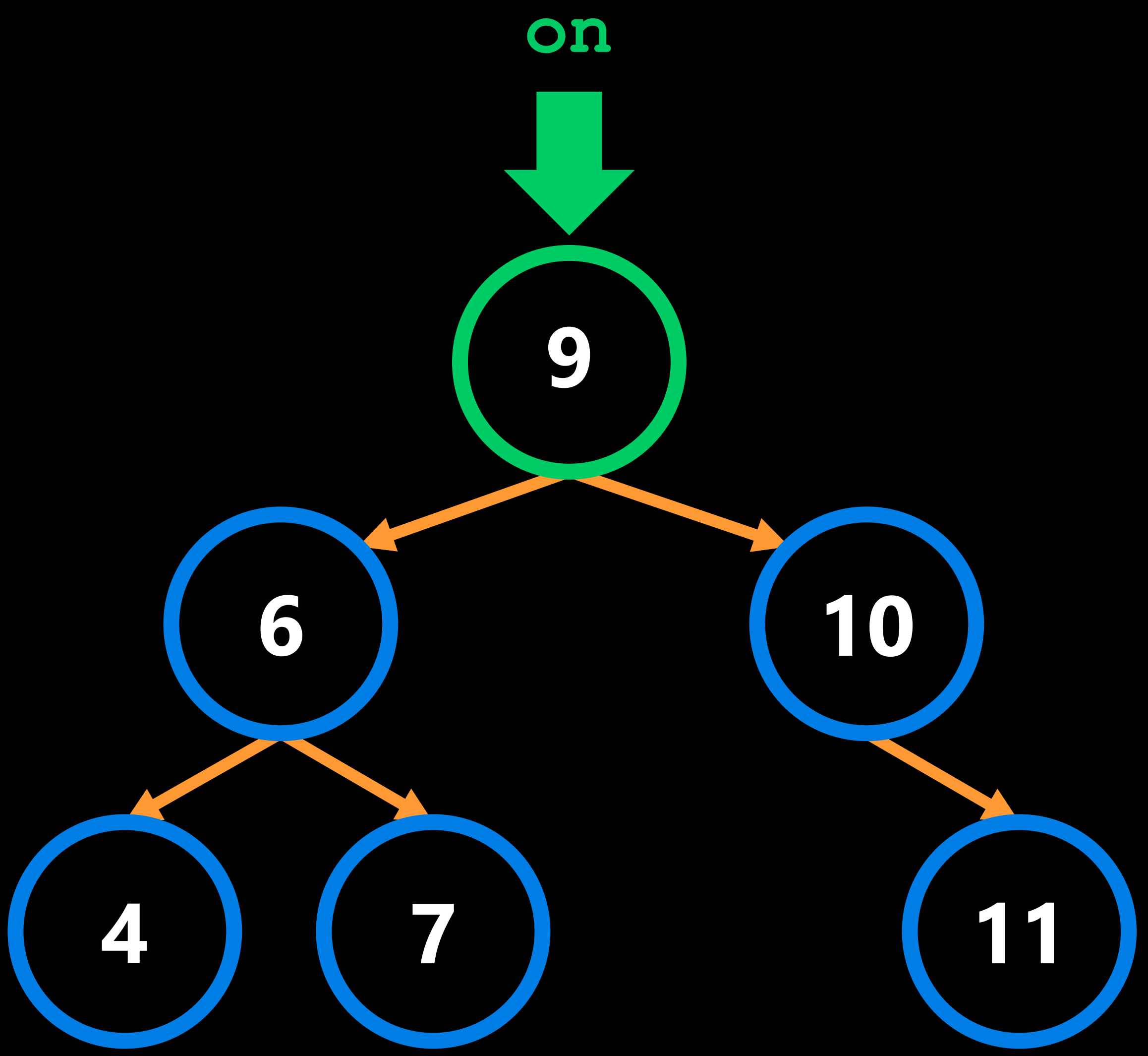
```python
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

True

True

Add on to stack.

stack = [9]



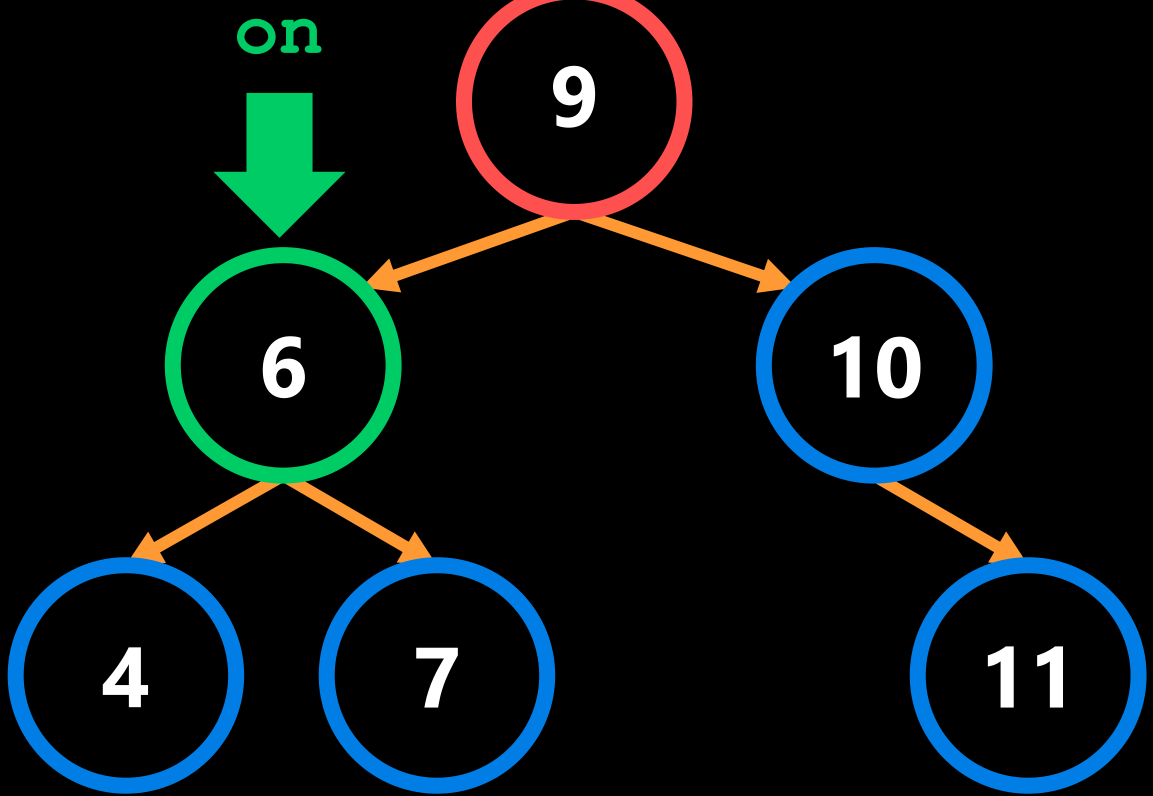
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

True

True

Move on to left node pointer.

stack = [9]

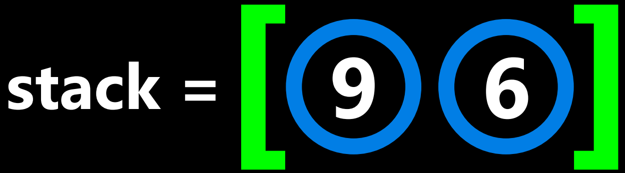


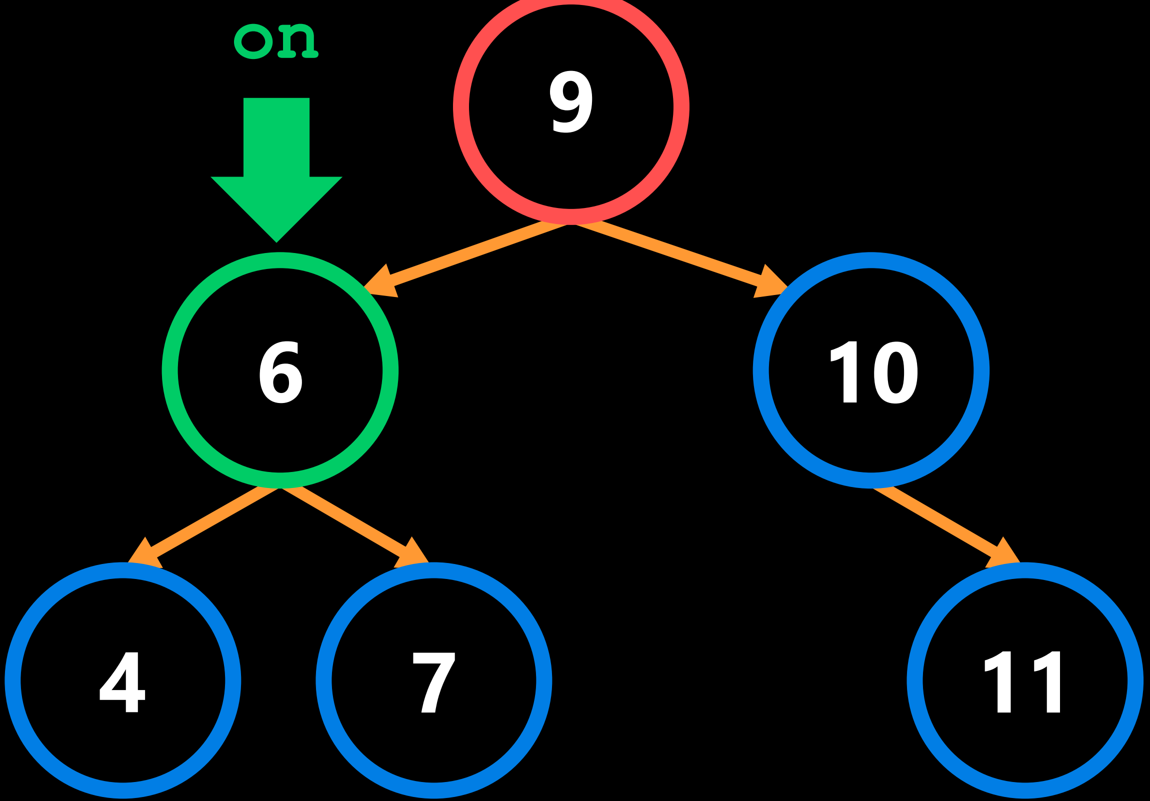
```python
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

True

True

Add on to stack.





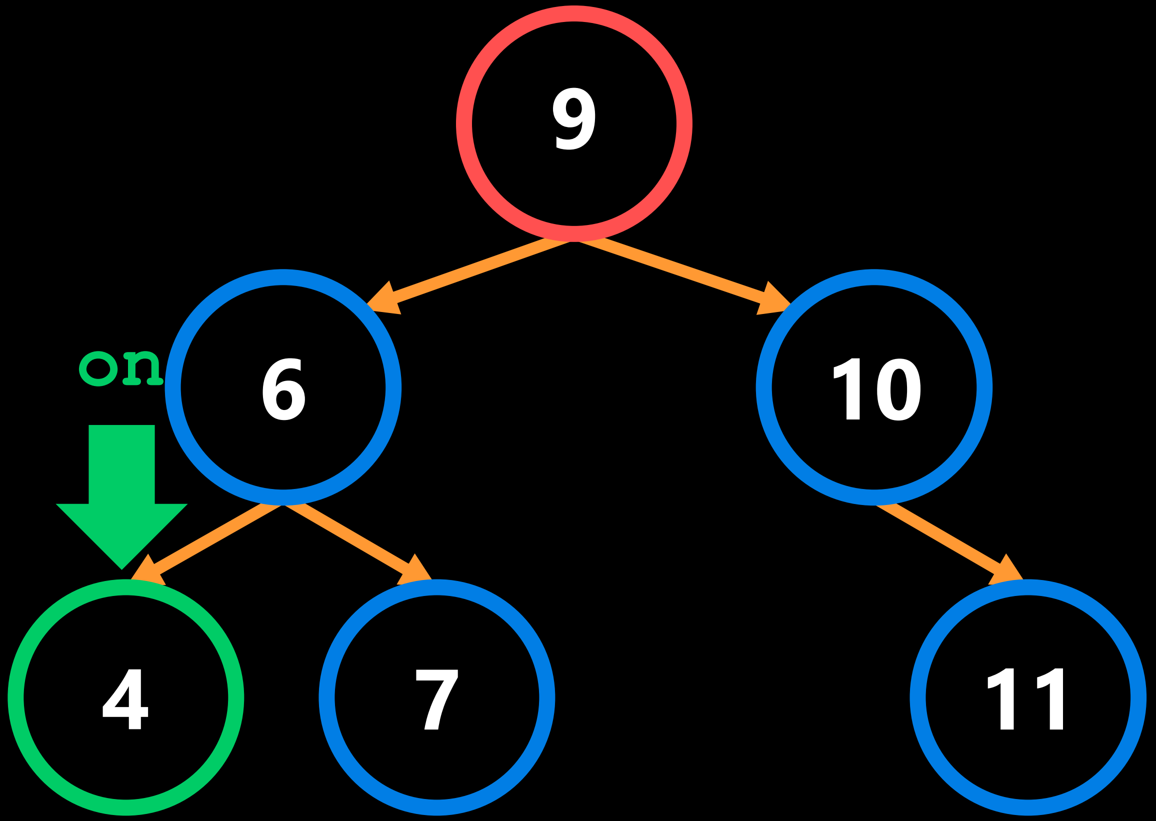
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

True

True

**Move on to left node pointer.**

**stack = [ 9 6 ]**



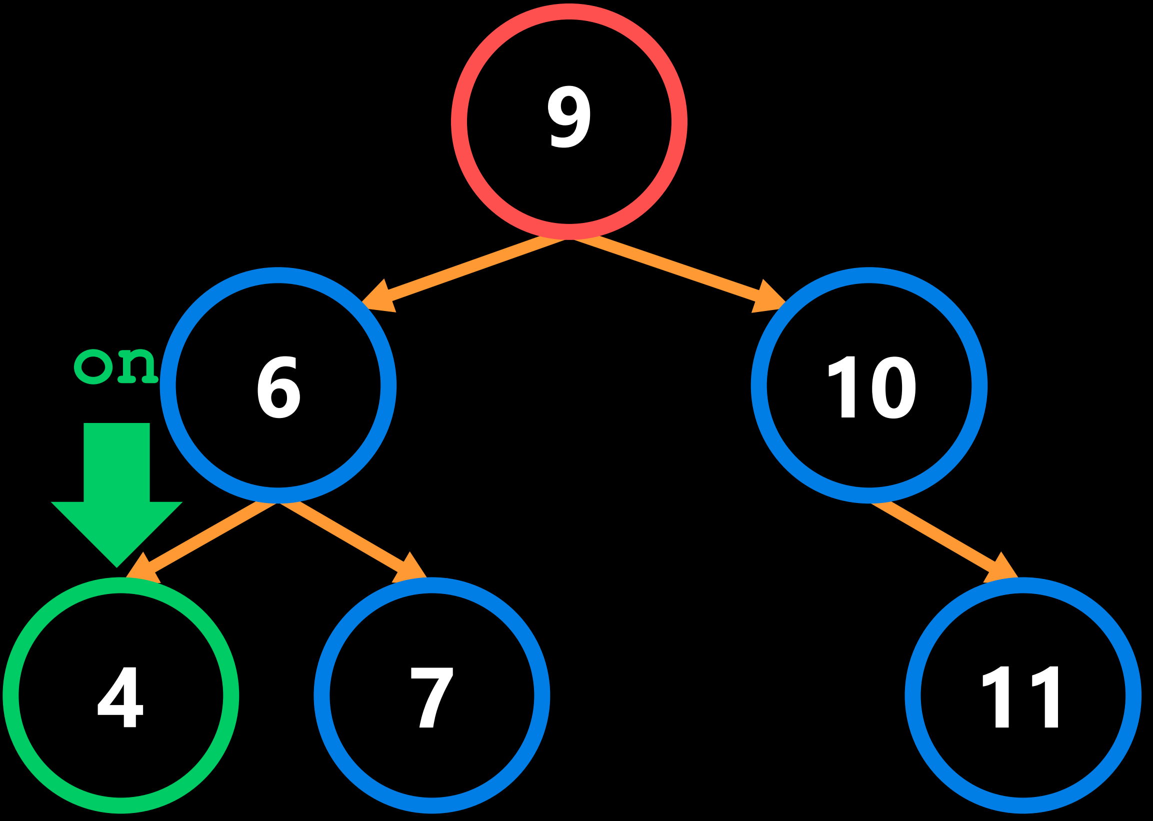
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

True

True

Add on to stack.

stack = [9 6 4]



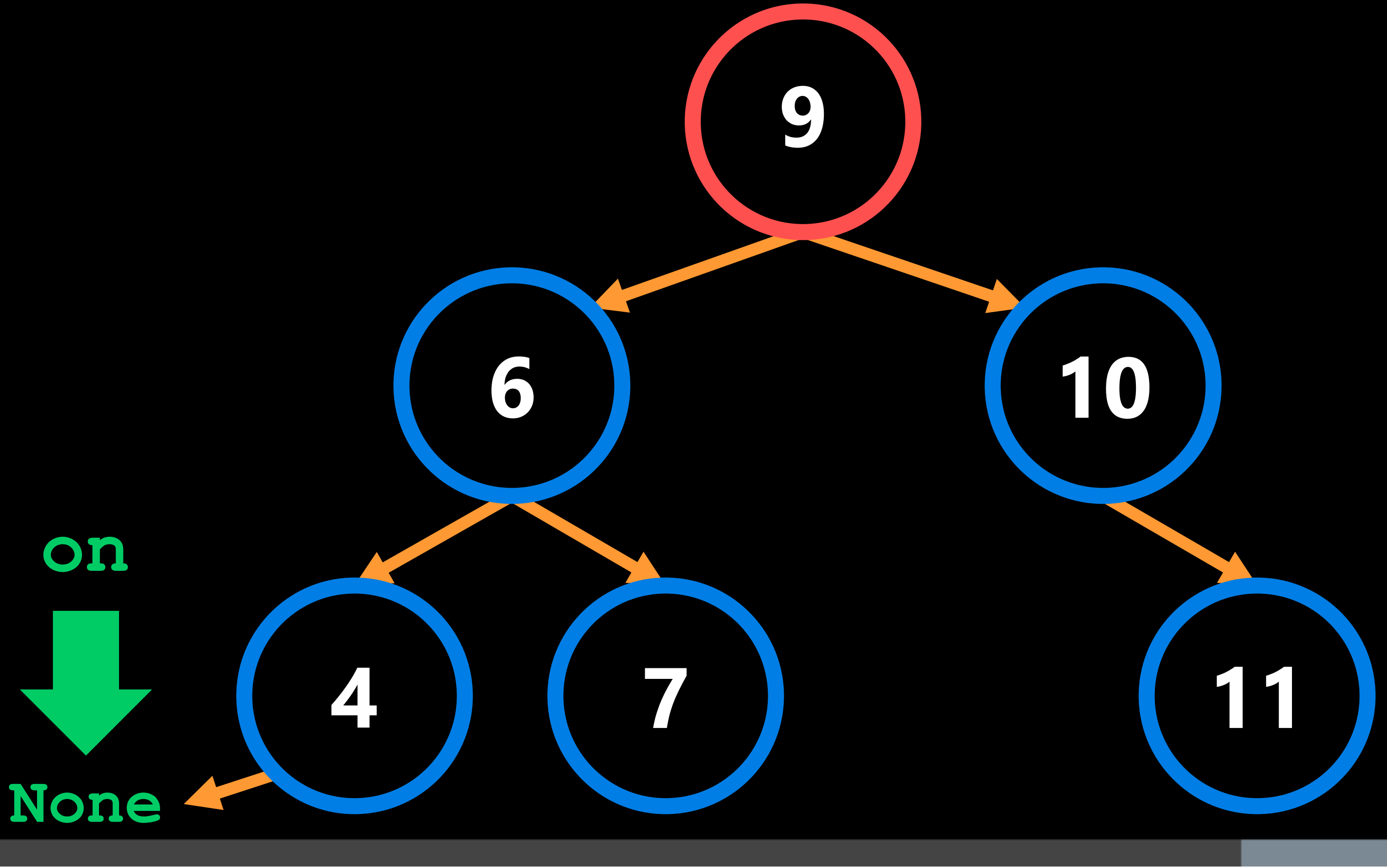
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

**stack = [9 6 4]**

True

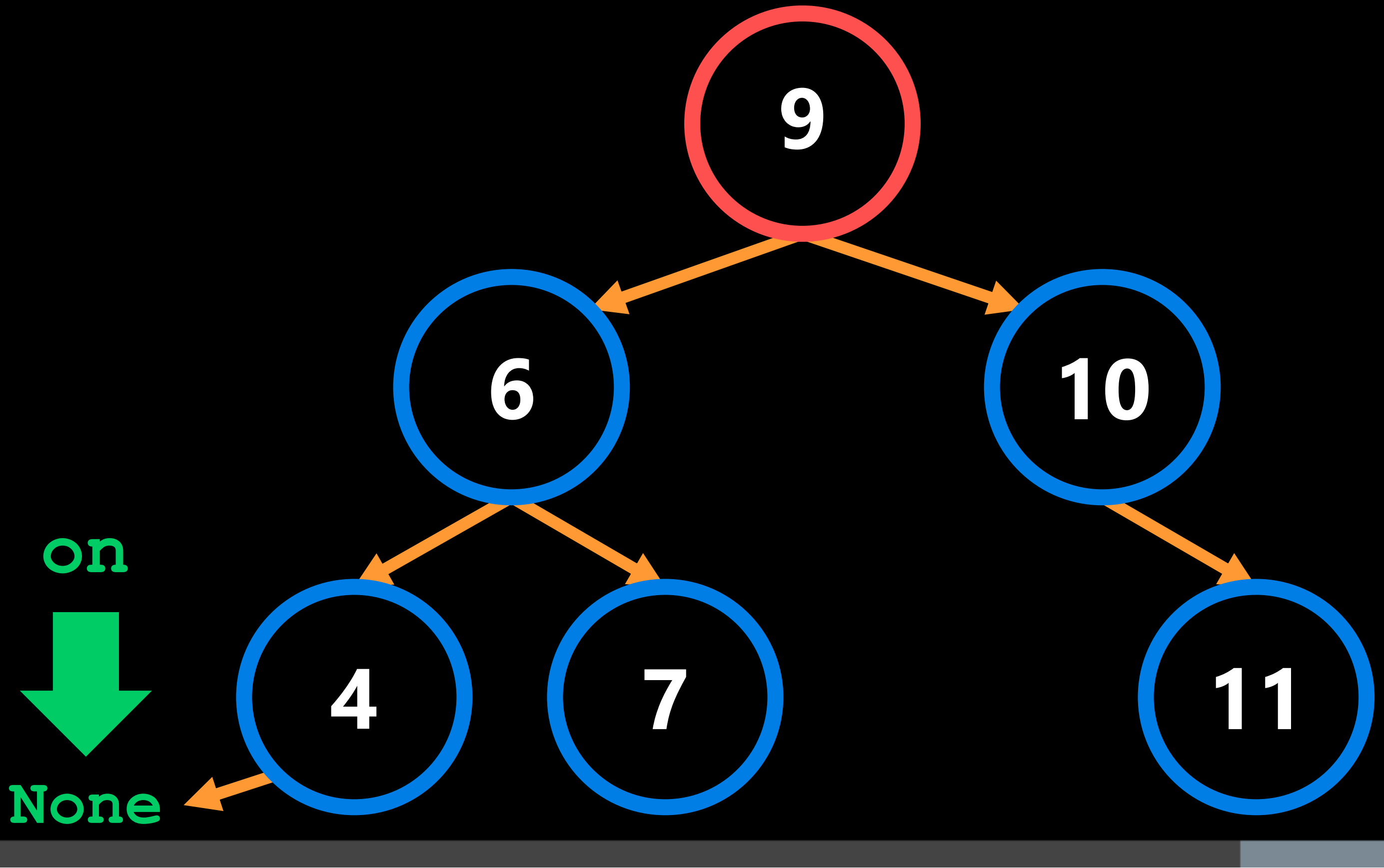
True

**Move on to left node pointer.**



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:  ⬅ True

            while on is not None:  ⬅ False
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

**stack = [9 6 4]**

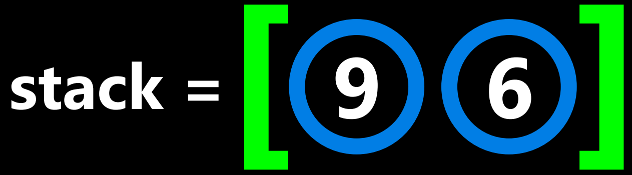


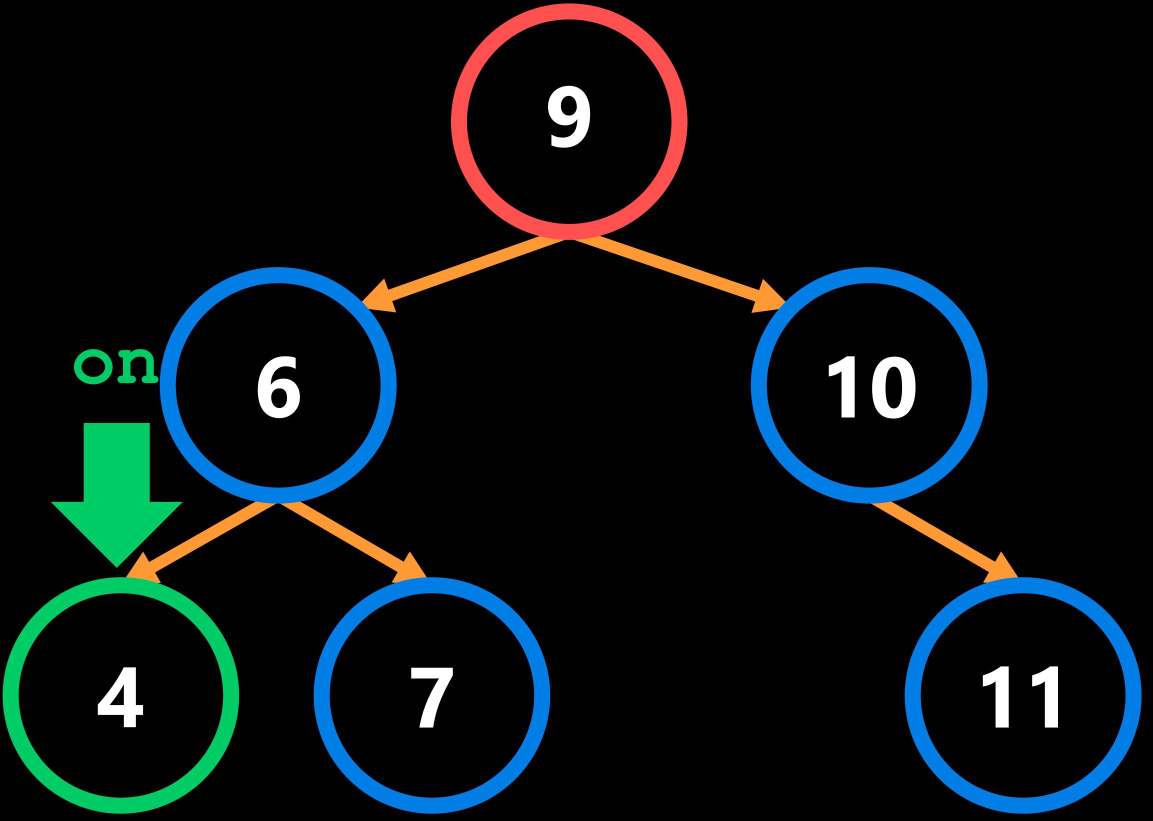
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

True

False

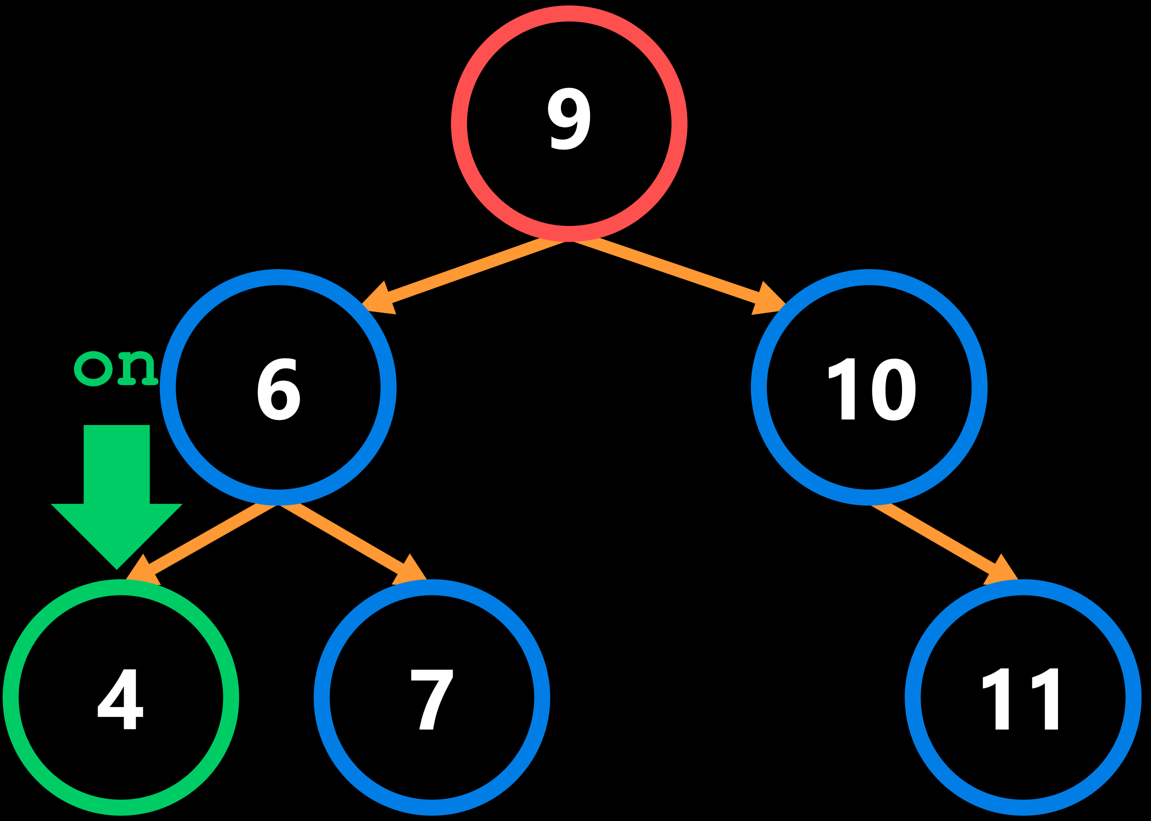
Set on to left node in stack.





```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:      True

            while on is not None:      False
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:      False
                return False

            prev = on
            on = on.right

        return True
```

**stack = [ 9 6 ]**



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

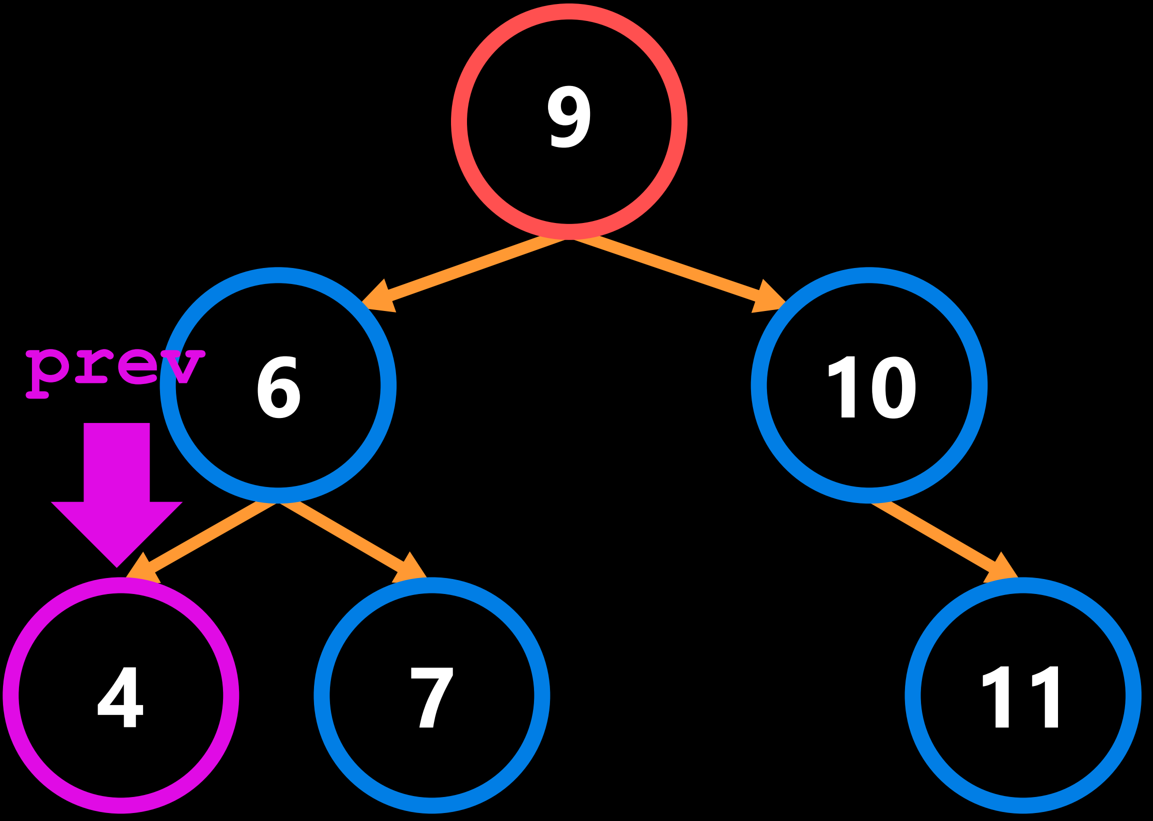
True

False

False

Set prev to on.

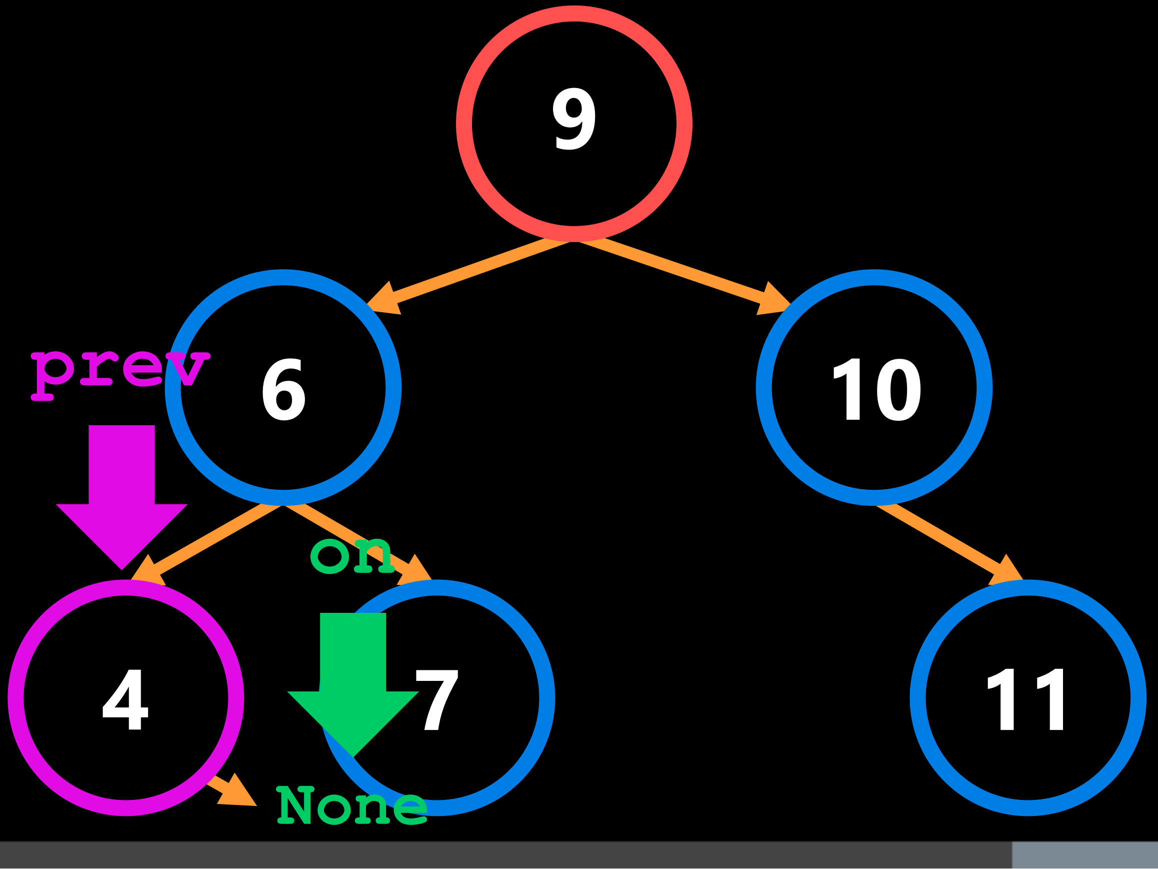
stack = [9 6]



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:  ← True

            while on is not None:  ← False
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:  ← False
                return False

            prev = on
            on = on.right  ←

        return True
```

**stack = [ 9 6 ]**

**Move on to the right pointer.**



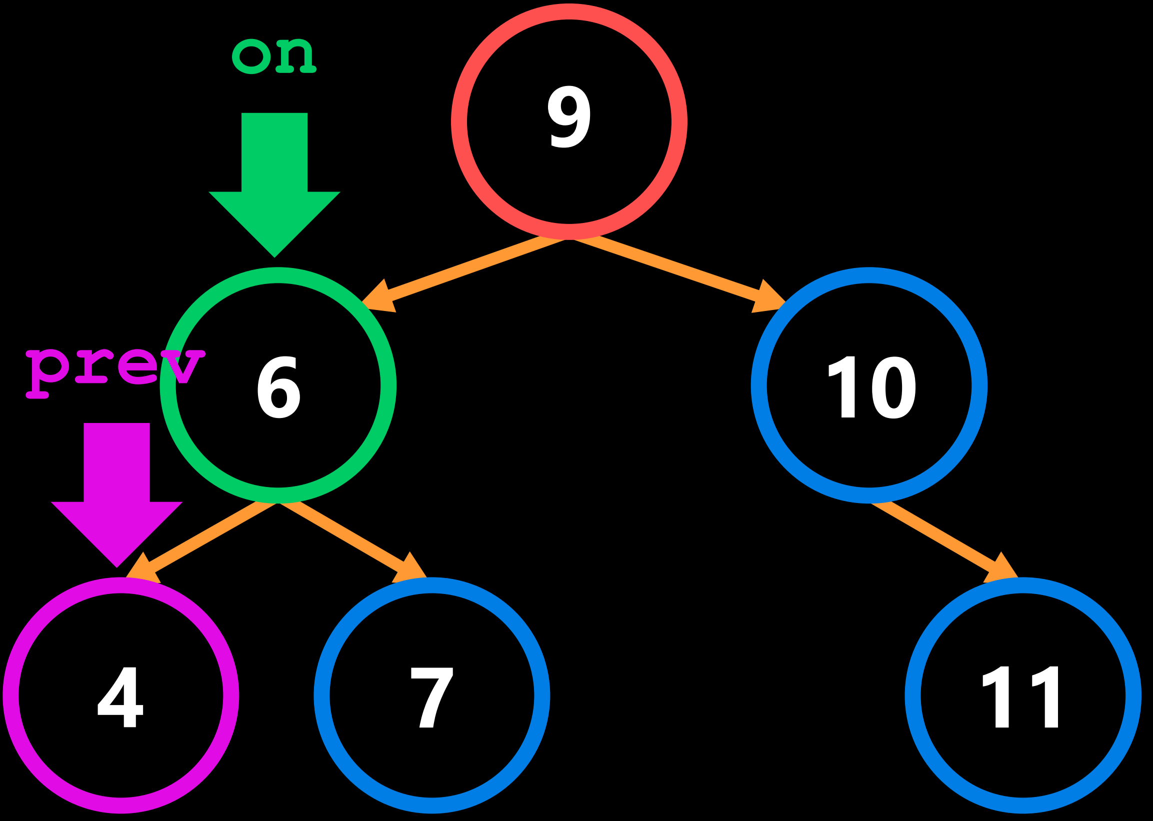
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

stack = [9]

True

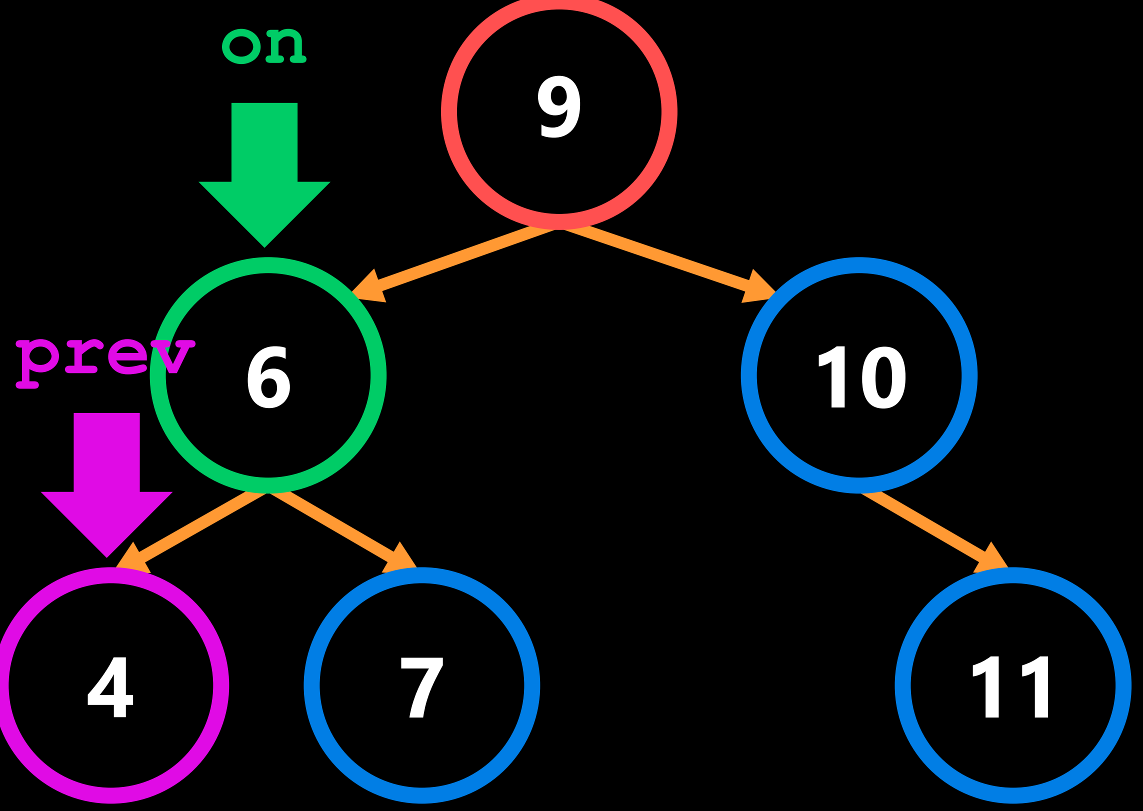
False

Set on to left node in stack.



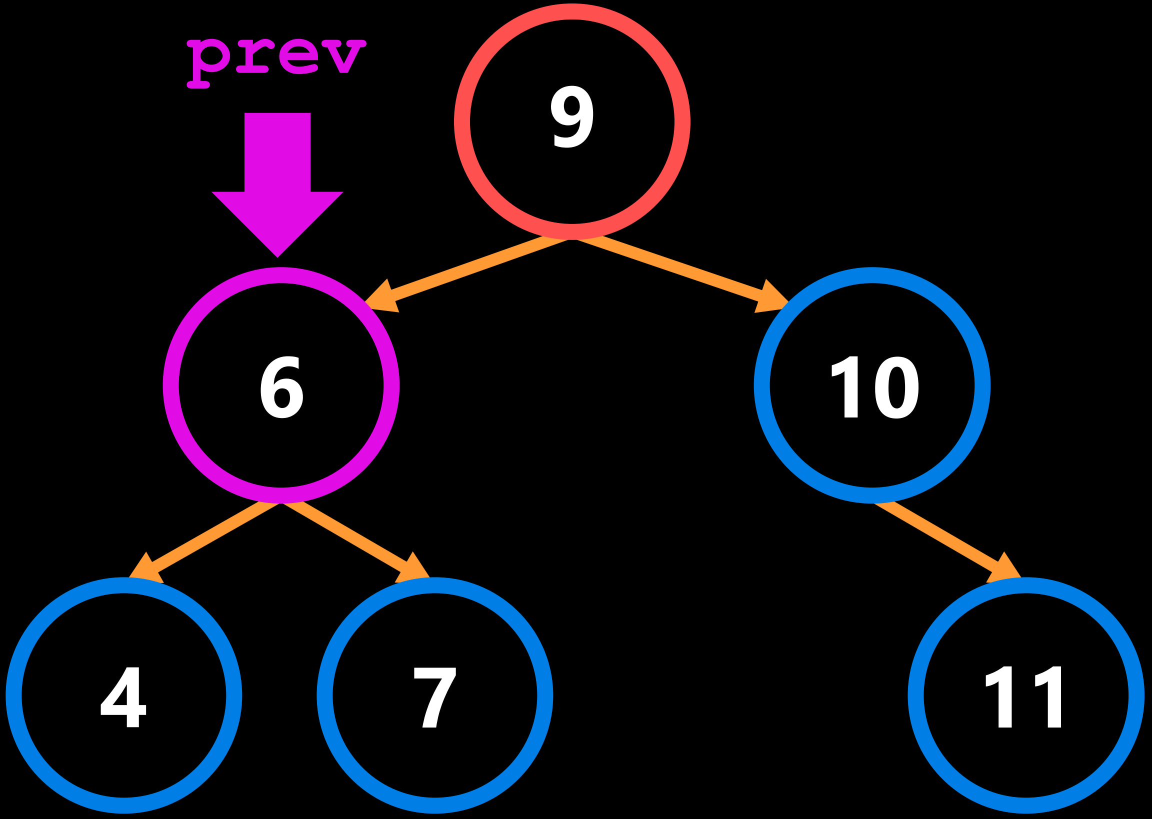
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:     True

            while on is not None:     False
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:     False
                return False

            prev = on
            on = on.right

        return True
```

stack = [9]



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:  ← True

            while on is not None:  ← False
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:  ← False
                return False

            prev = on  ← Set prev to on.
            on = on.right

        return True
```

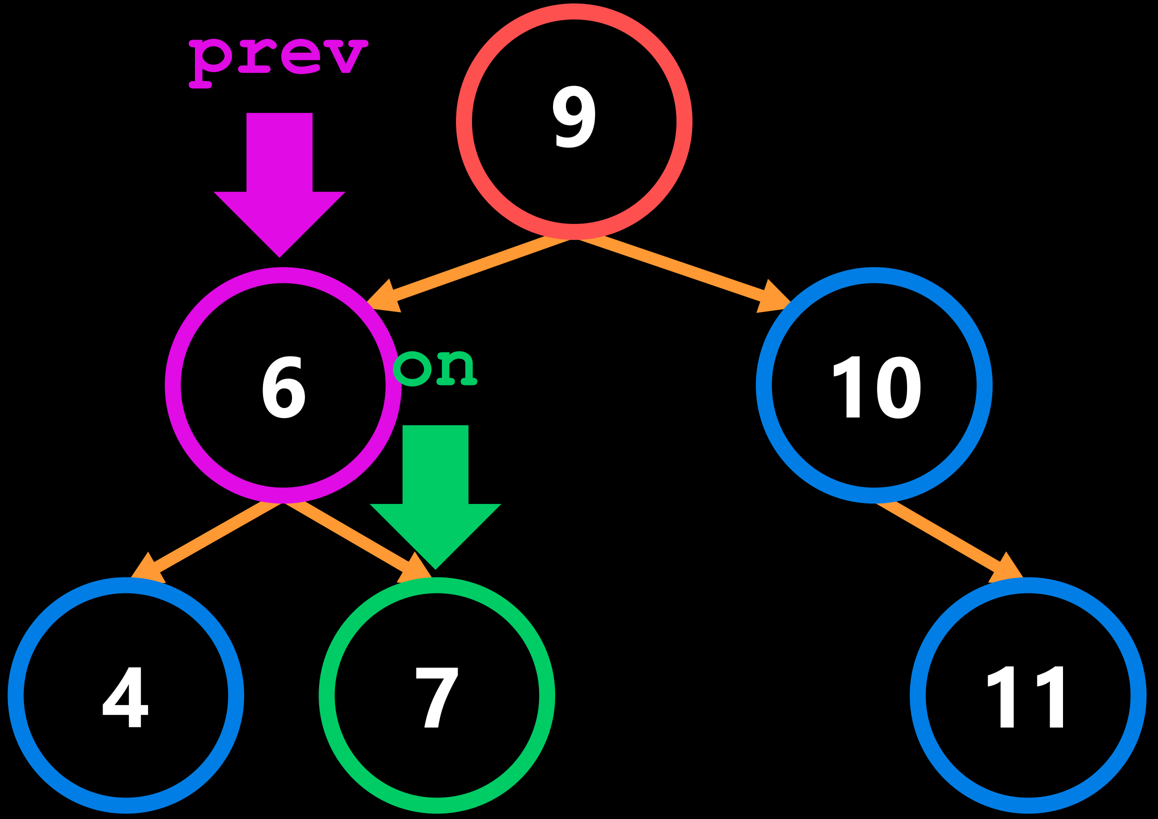
stack = [9]



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:  ← True

            while on is not None:  ← False
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:  ← False
                return False

            prev = on
            on = on.right  ←

        return True
```

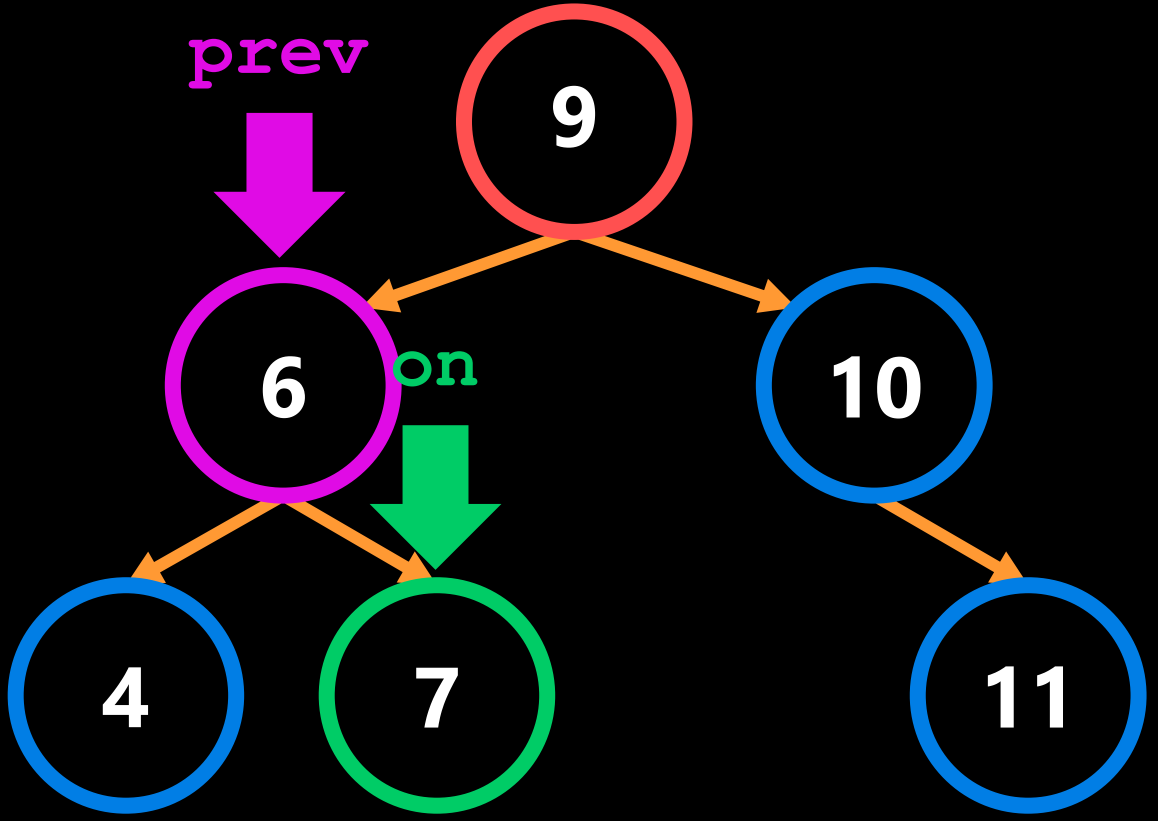
Move on to the right pointer.

stack = [9]



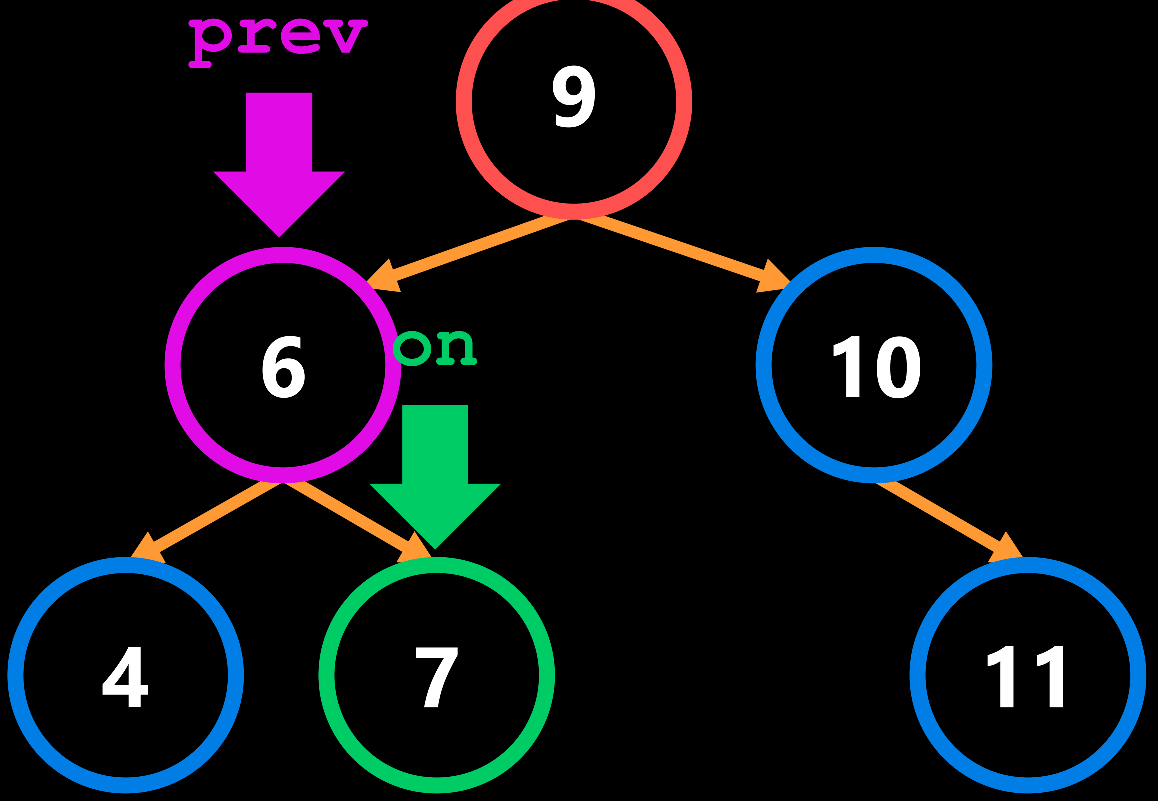
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:   True

            while on is not None:   True
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

stack = [9]



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:  ⬅ True

            while on is not None:  ⬅ True
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

stack = [ 9 ]



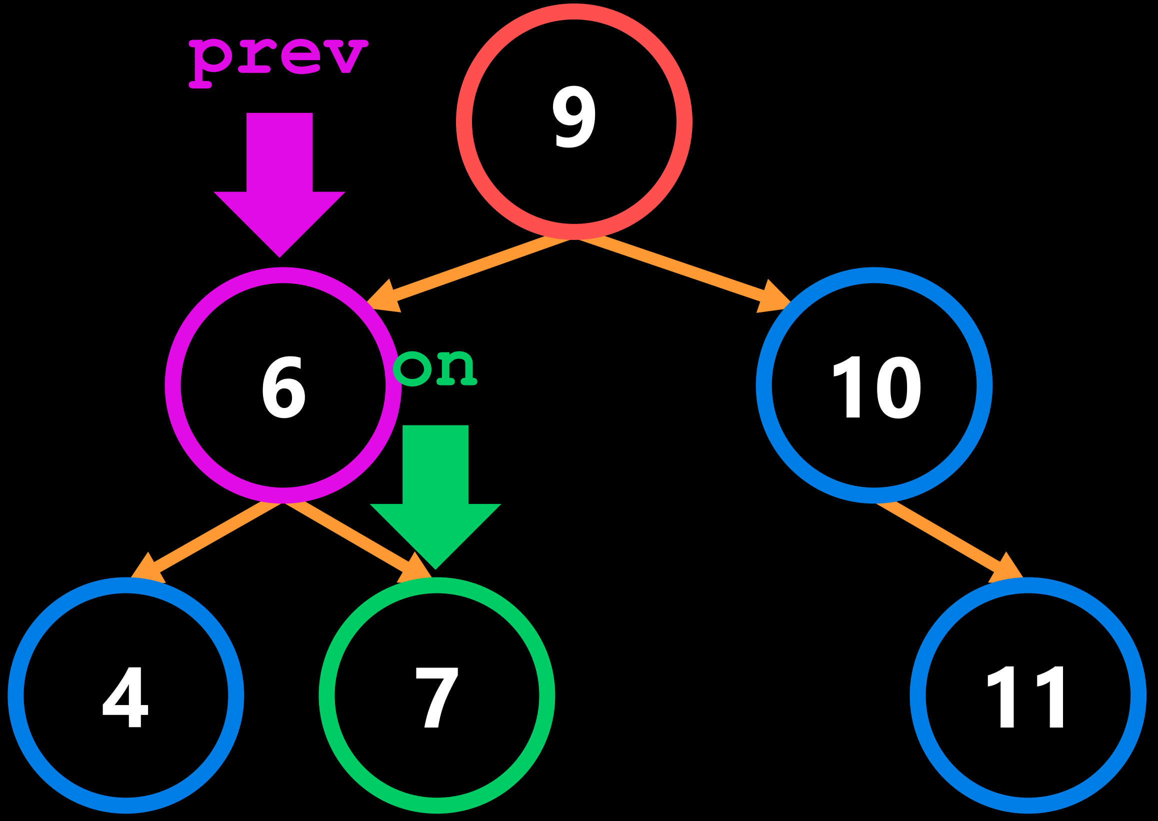
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

True

True

Add on to stack.

stack = [9 7]



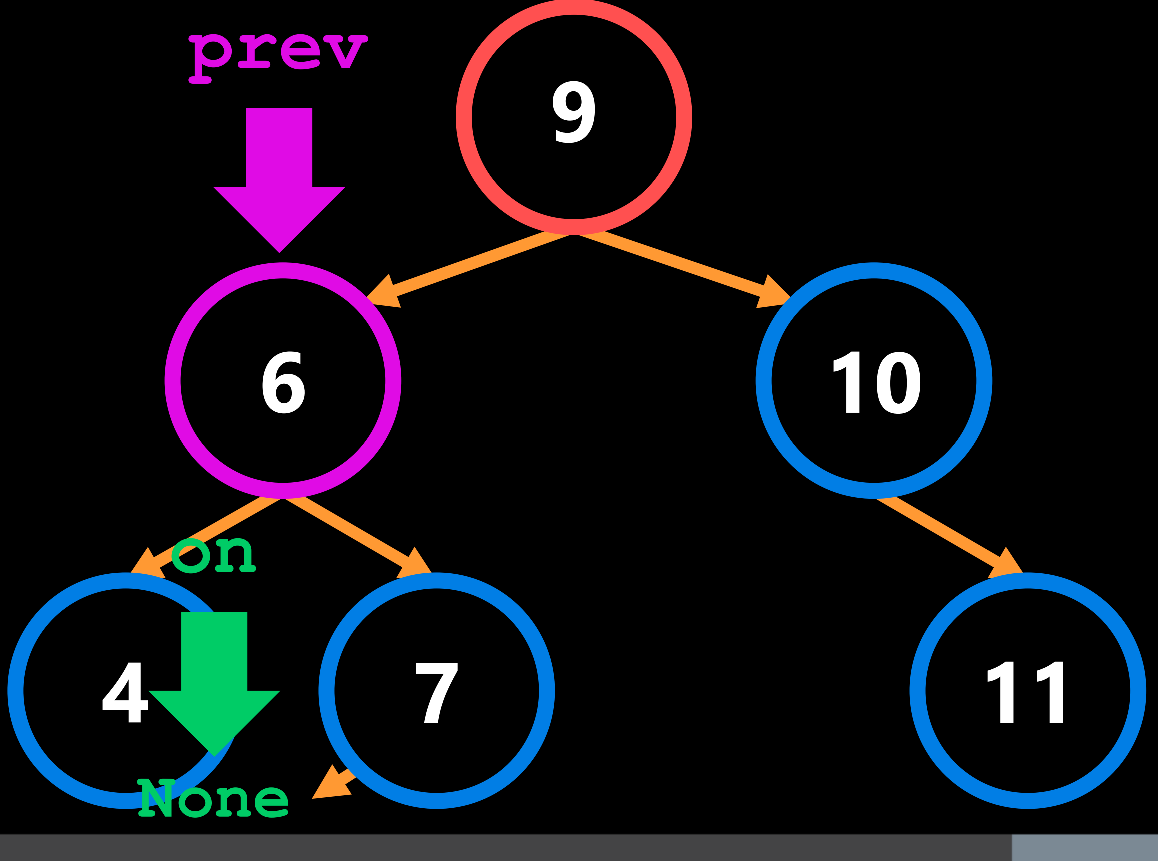
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

stack = [9 7]

True

True

**Move on to left node pointer.**



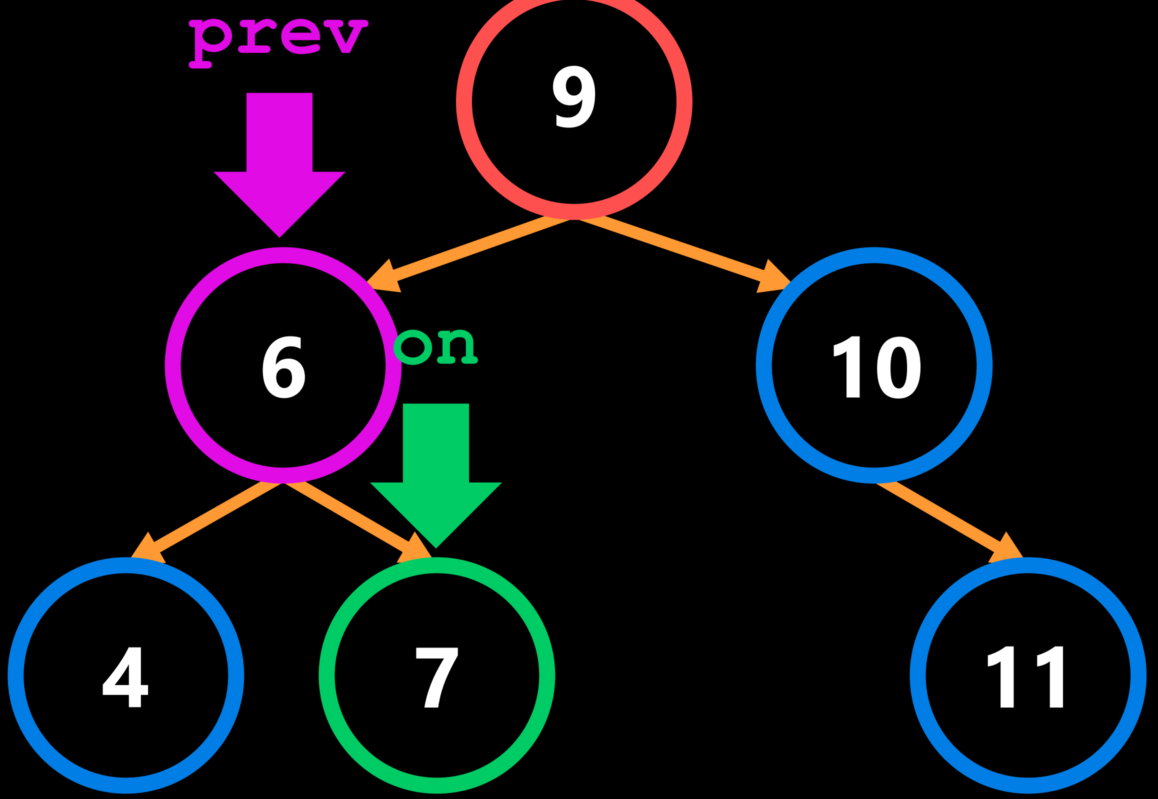
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

stack = [9]

while len(stack) > 0 or on is not None: ← True

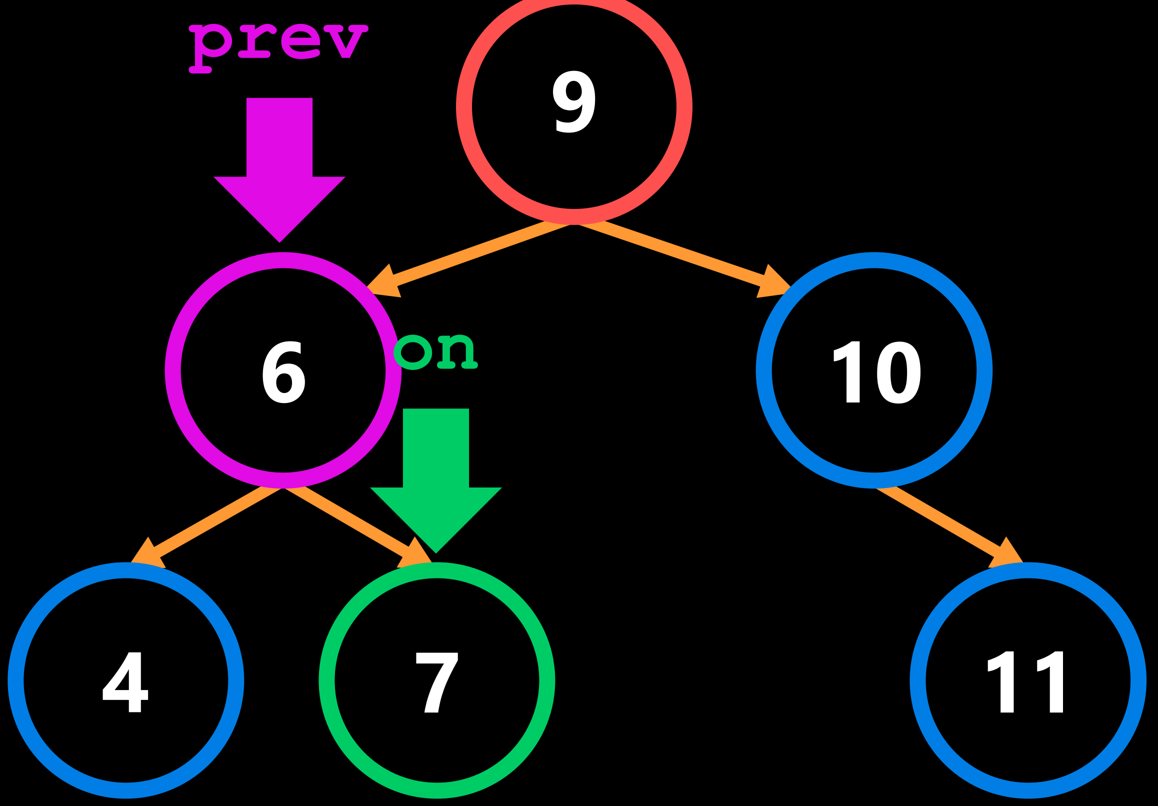
while on is not None: ← True

on = stack.pop() ← Set on to left node in stack.



```python
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:  ← True

            while on is not None:  ← True
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:  ← False
                return False

            prev = on
            on = on.right

        return True
```

stack = [ 9 ]



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

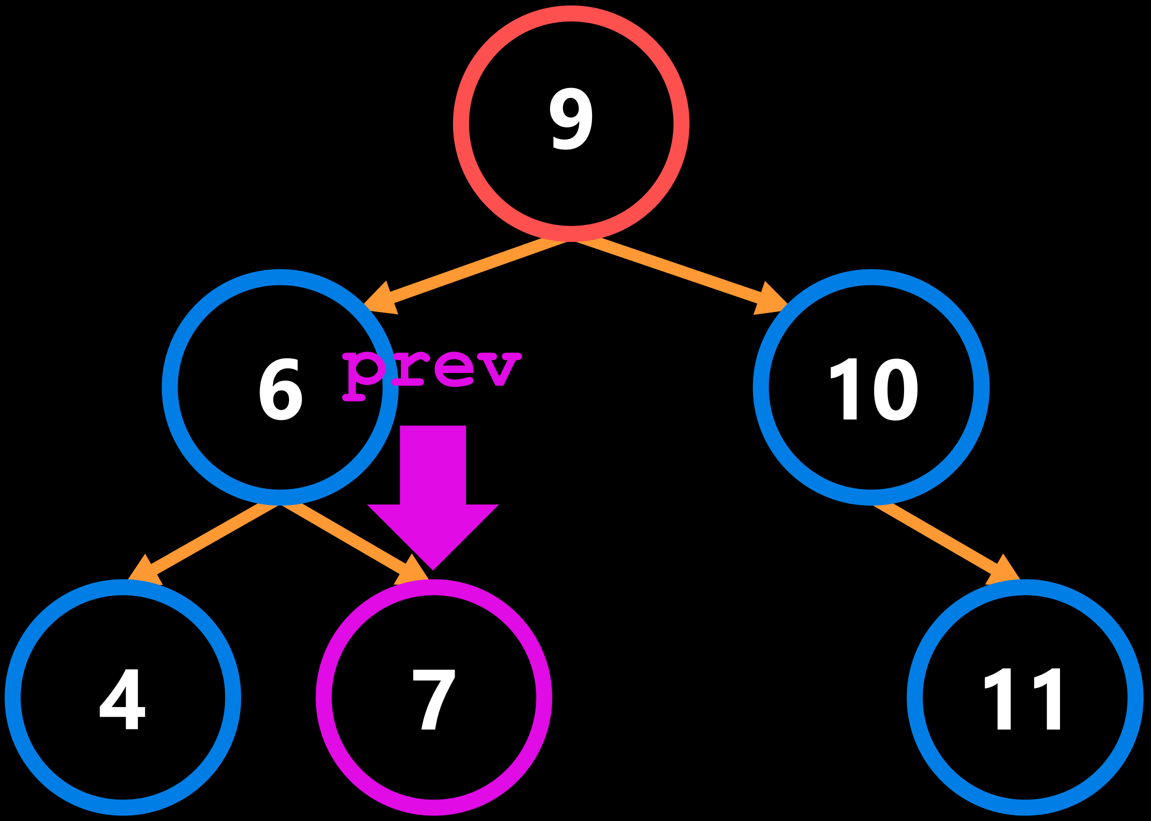
while len(stack) > 0 or on is not None: ← True

while on is not None: ← True

if prev is not None and on.cargo <= prev.cargo: ← False

prev = on ← Set prev to on.

stack = [9]



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

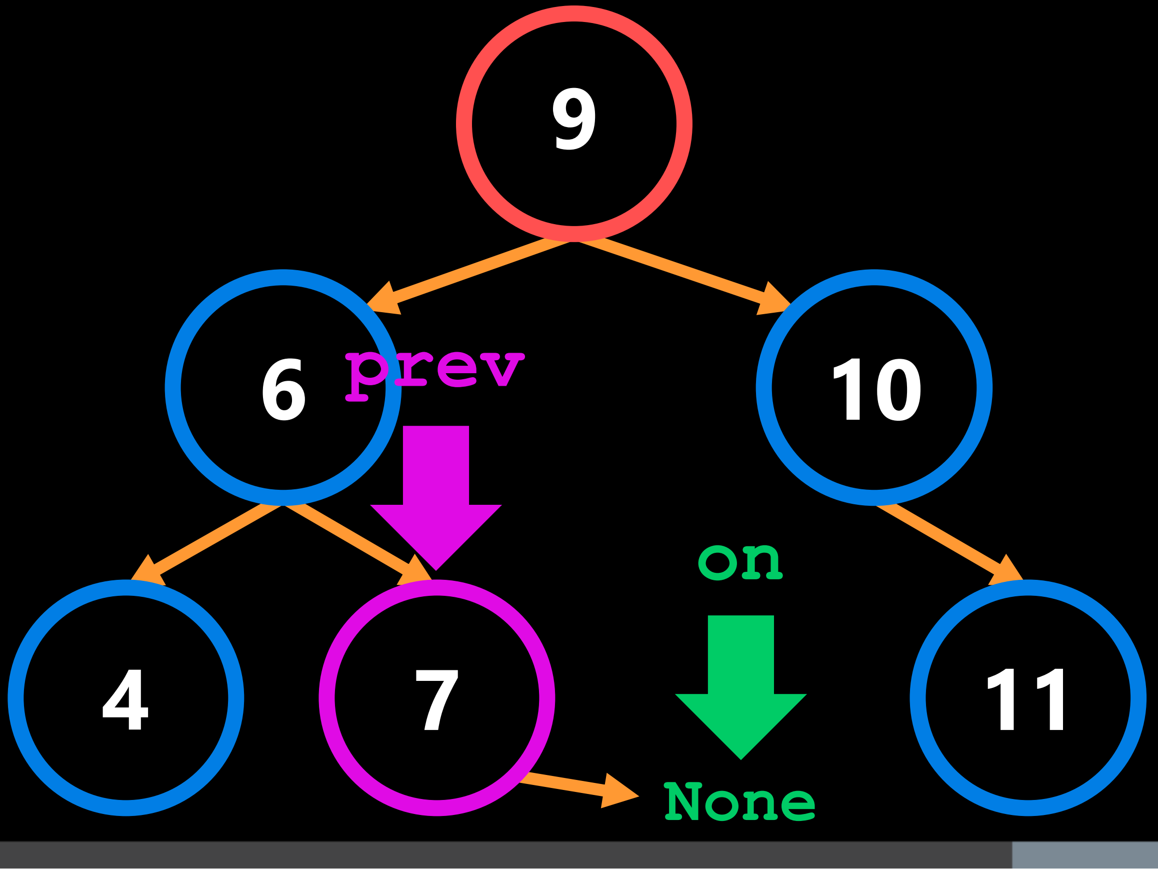
True

True

False

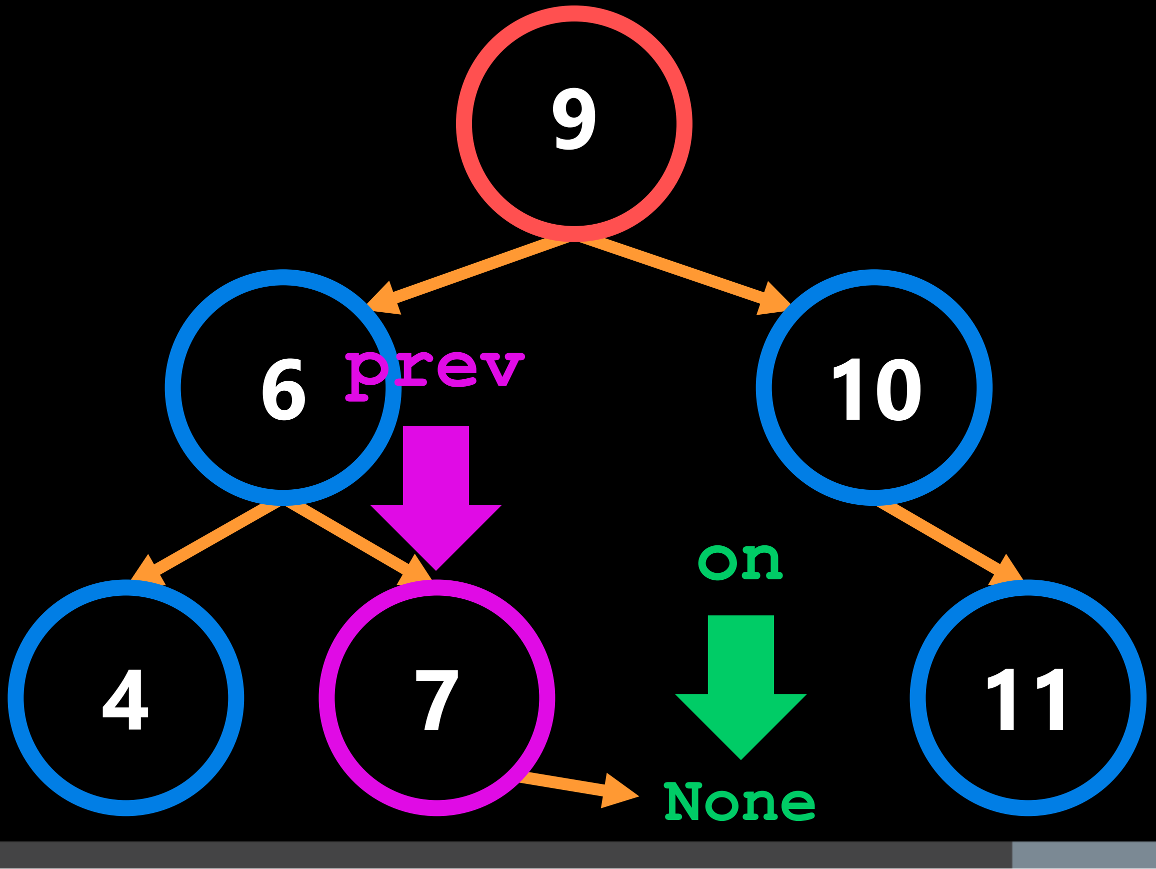
Move **on** to the right pointer.

stack = [9]



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:  ← True

            while on is not None:  ← False
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

**stack = [9]**



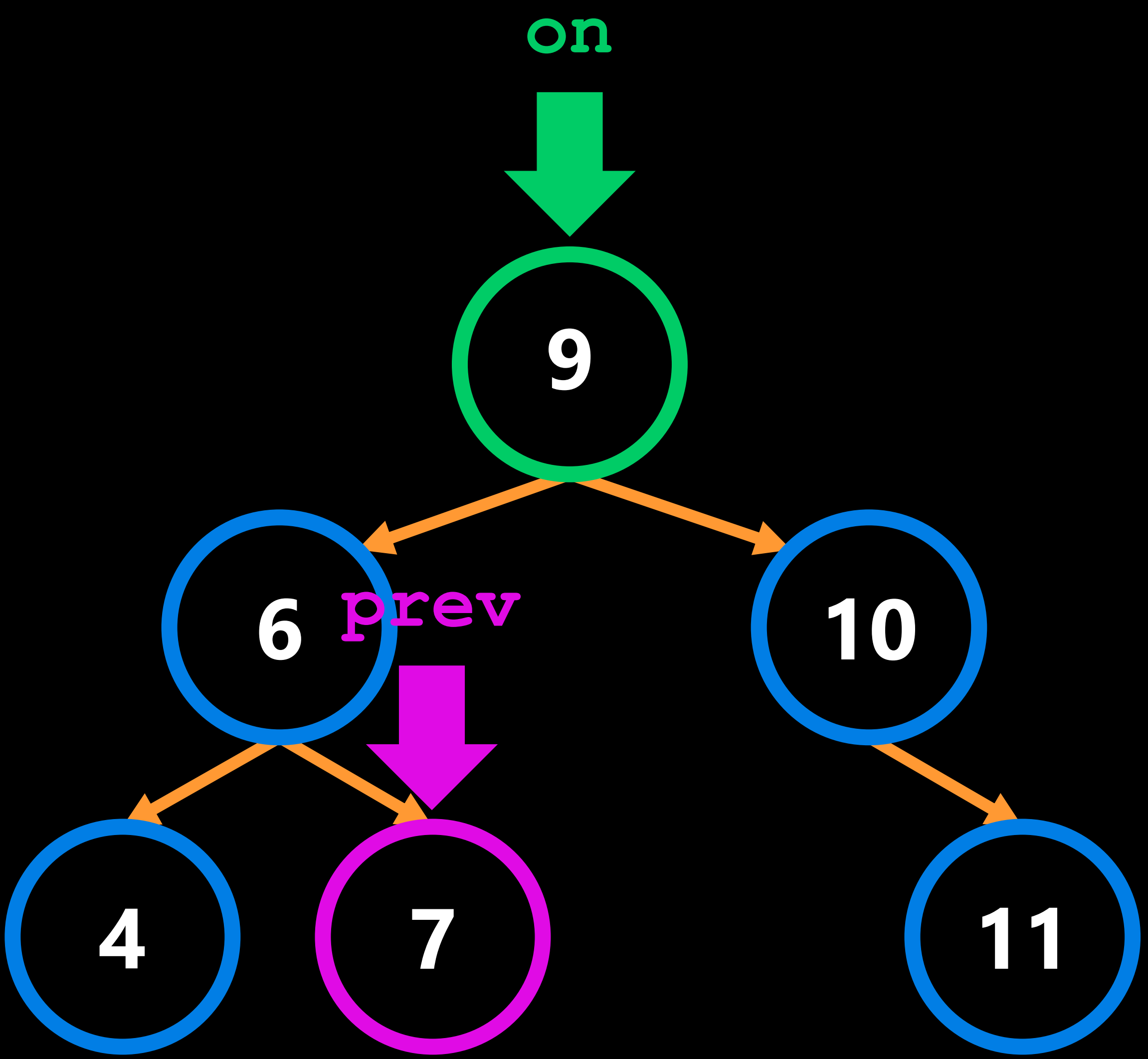
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

**stack = [ ]**

True

False

**Set on to left node in stack.**



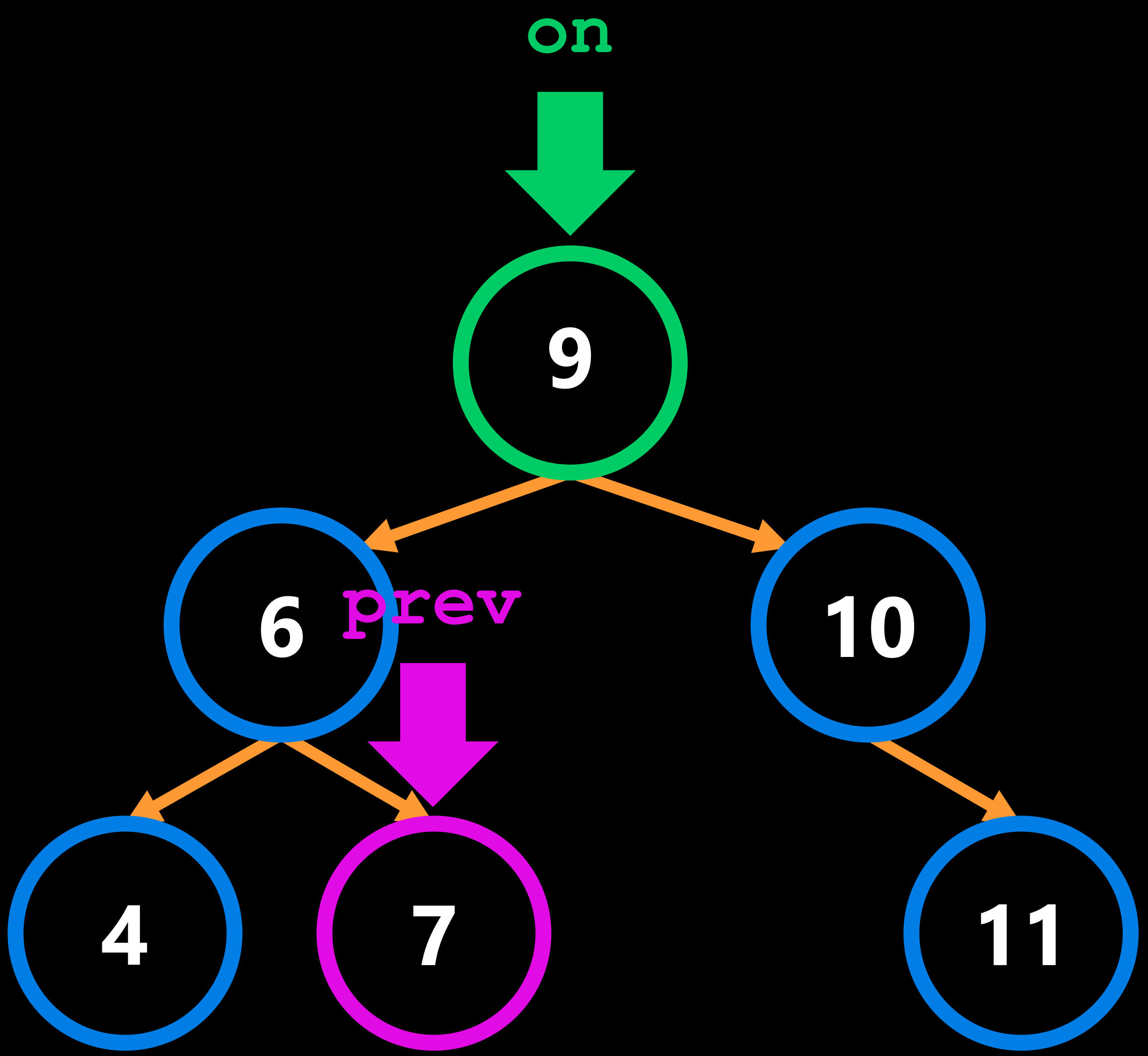
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

True

False

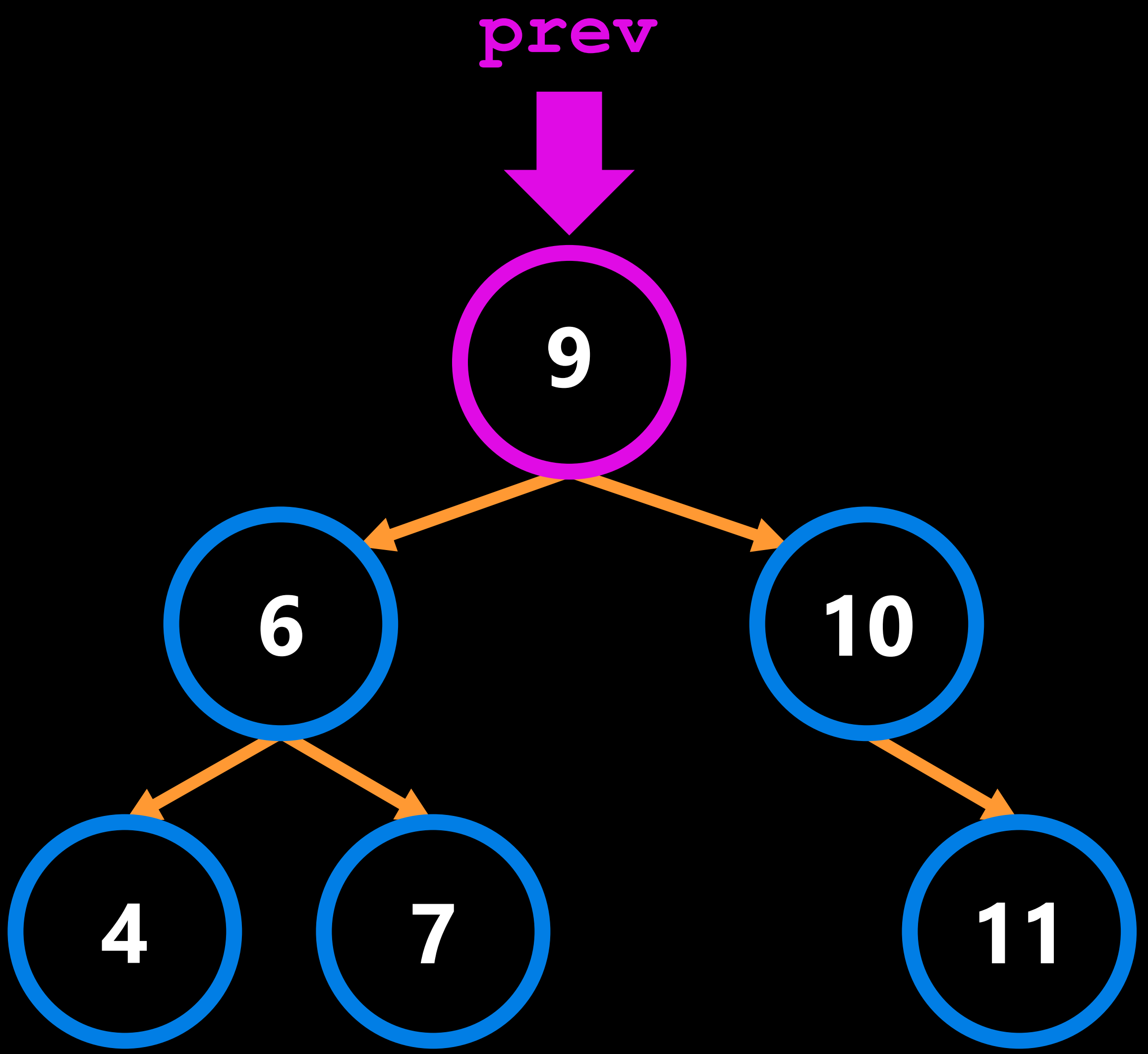
False

stack = [ ]



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:  ← True

            while on is not None:  ← False
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:  ← False
                return False

            prev = on  ← Set prev to on.
            on = on.right

        return True
```

**stack = [ ]**



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

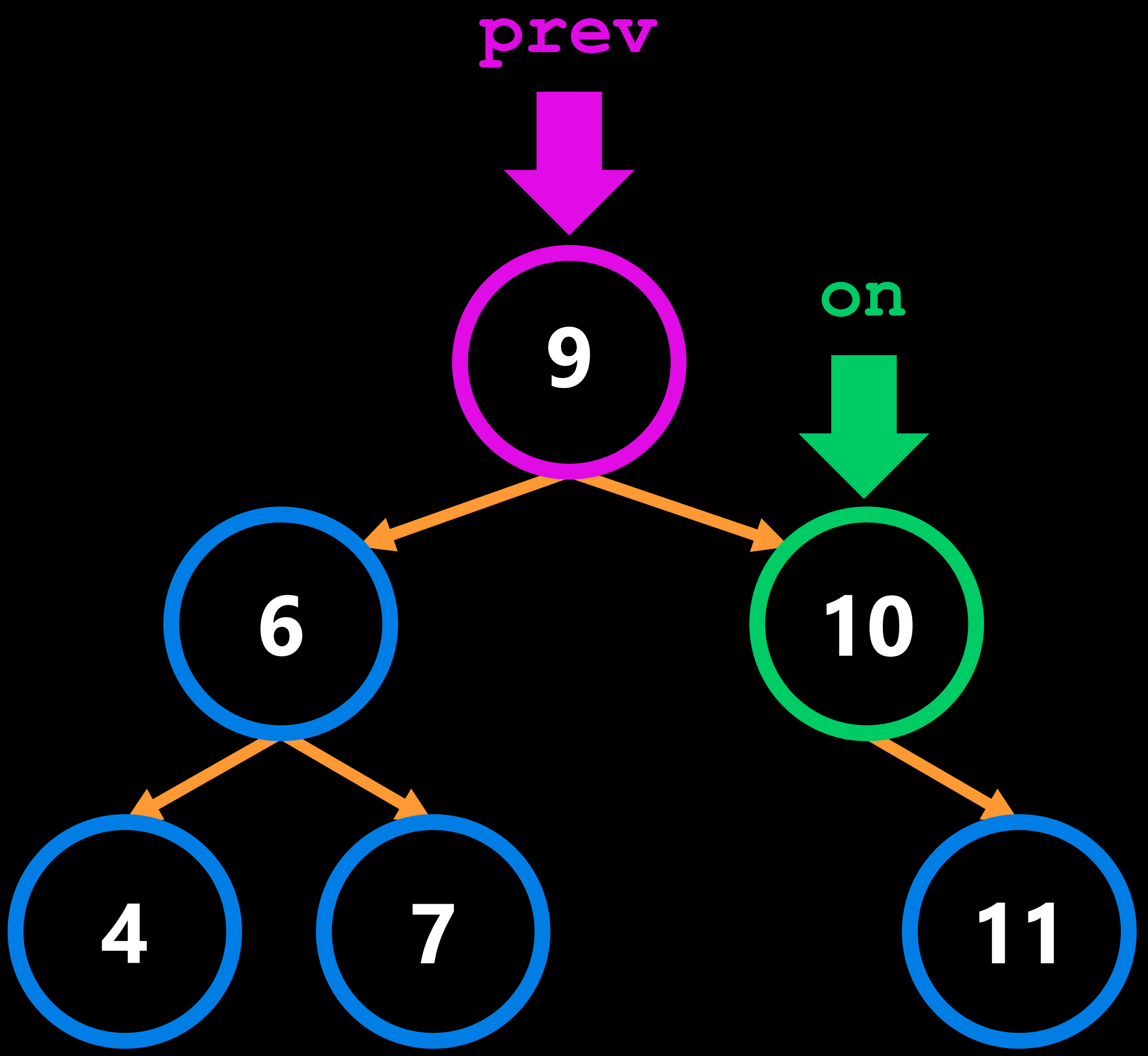
True

False

False

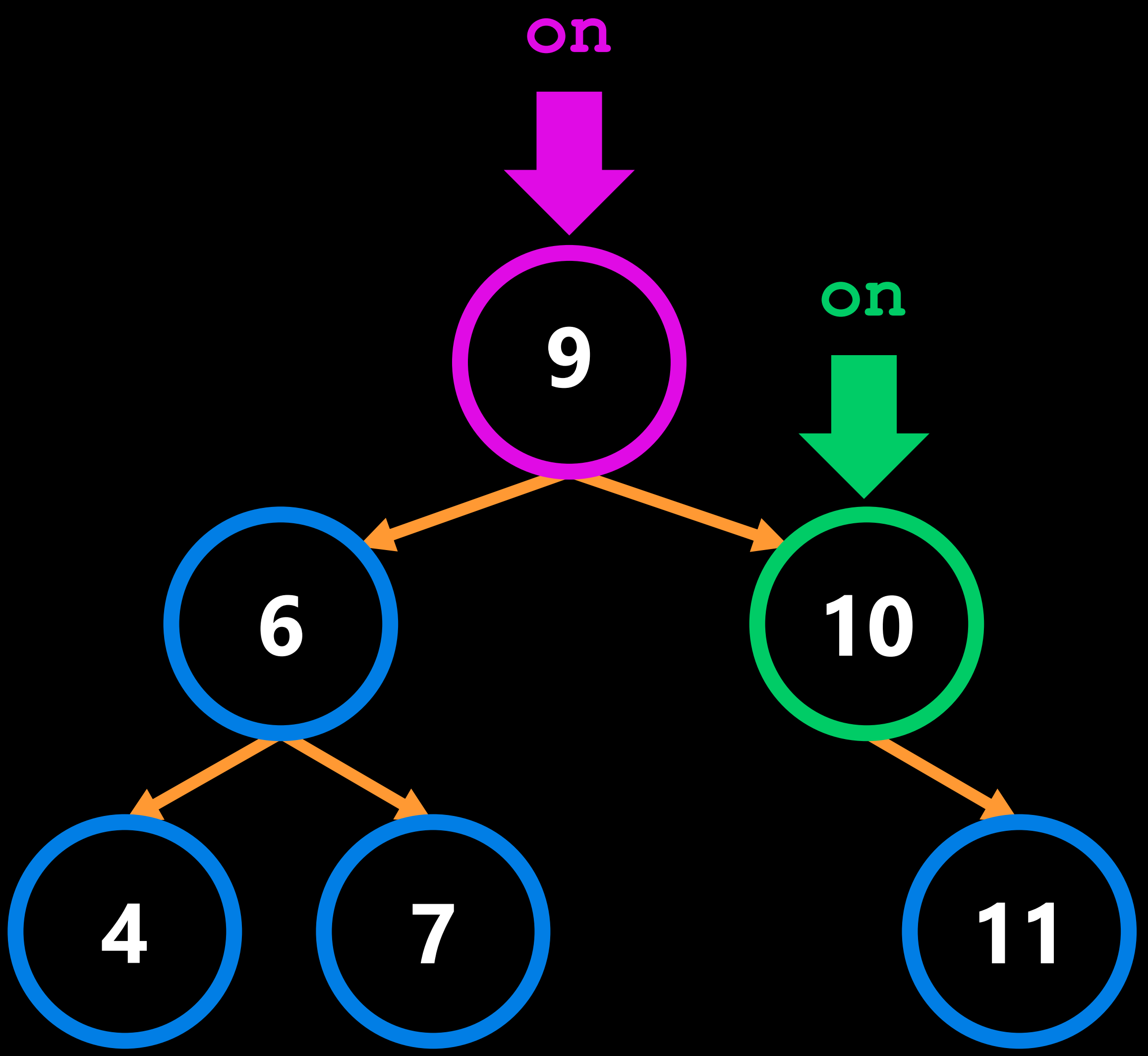
Move on to the right pointer.

stack = [ ]



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:  ⬅ True

            while on is not None:  ⬅ True
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

**stack = [ ]**



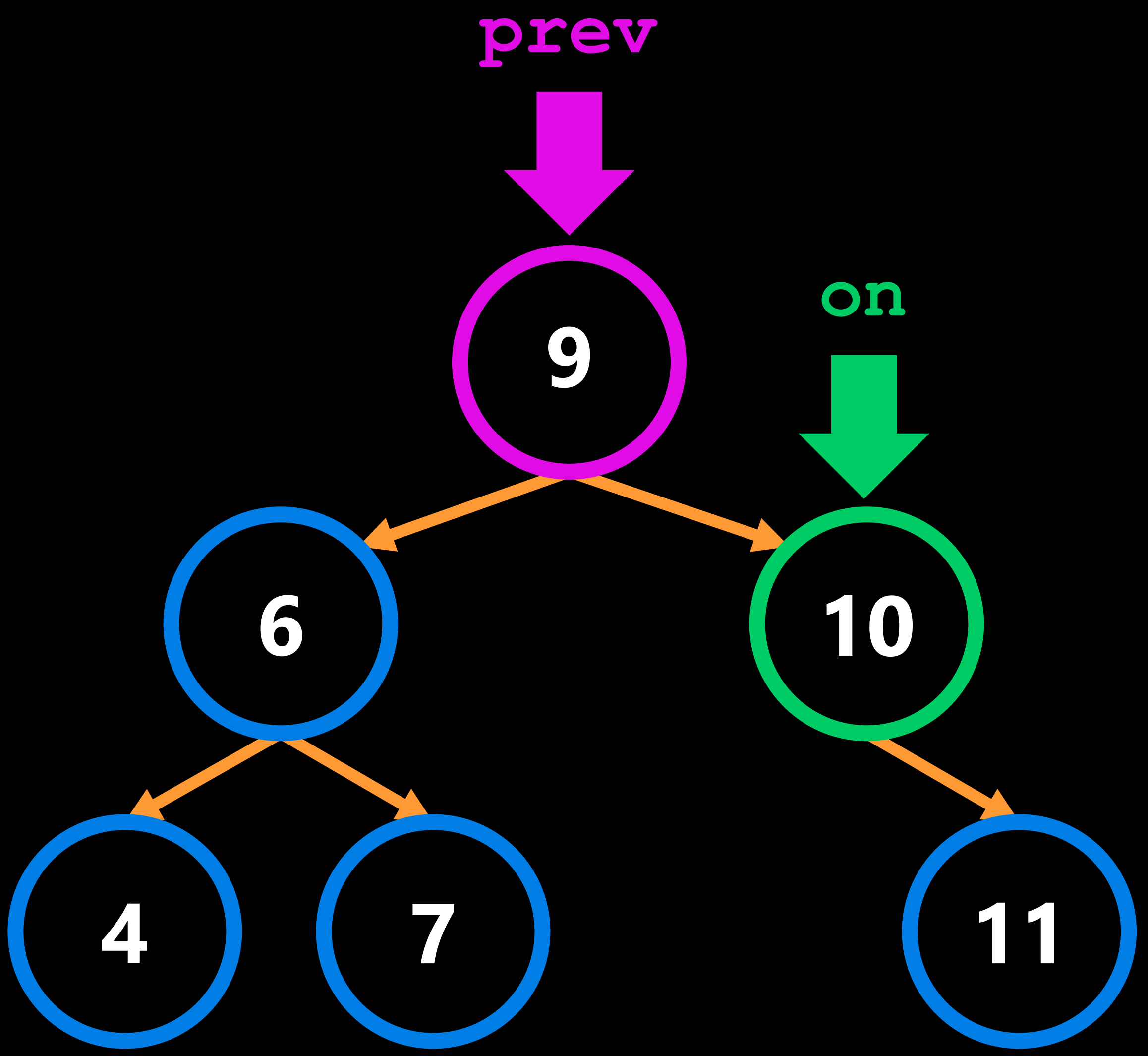
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

True

True

Add on to stack.

stack = [ 10 ]



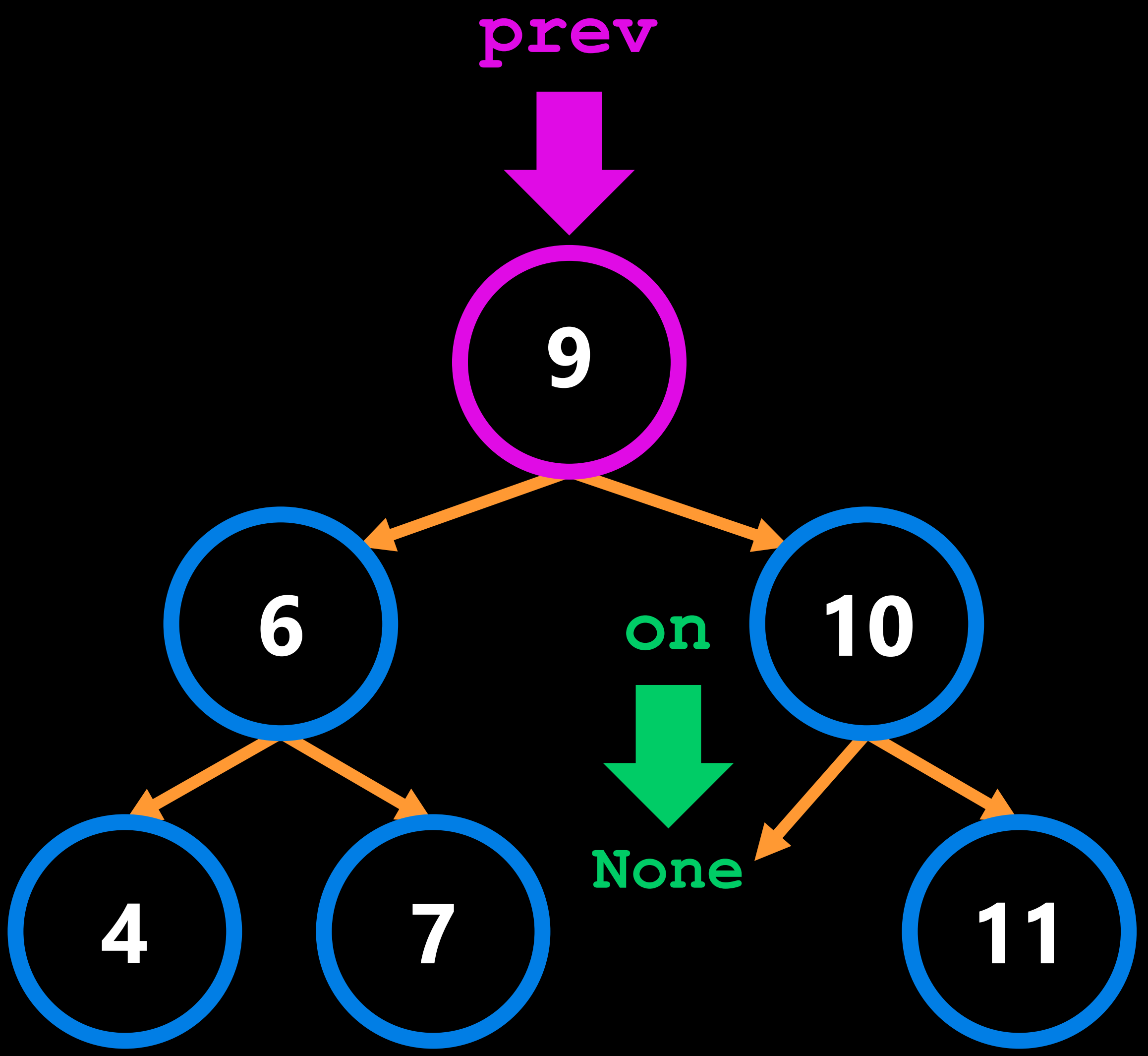
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

stack = [10]

True

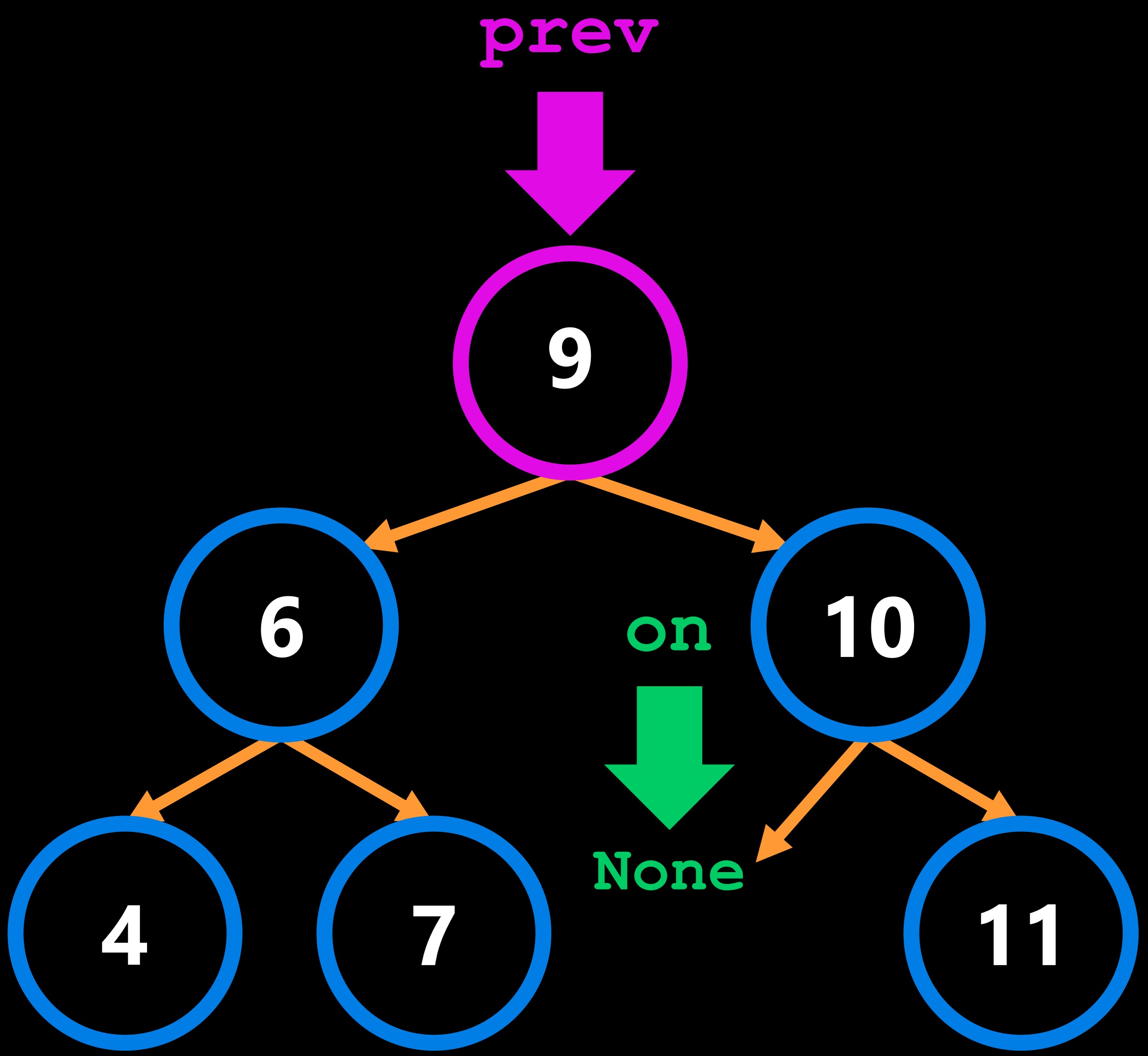
True

Move on to left node pointer.



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:  ← True

            while on is not None:  ← False
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

stack = [10]



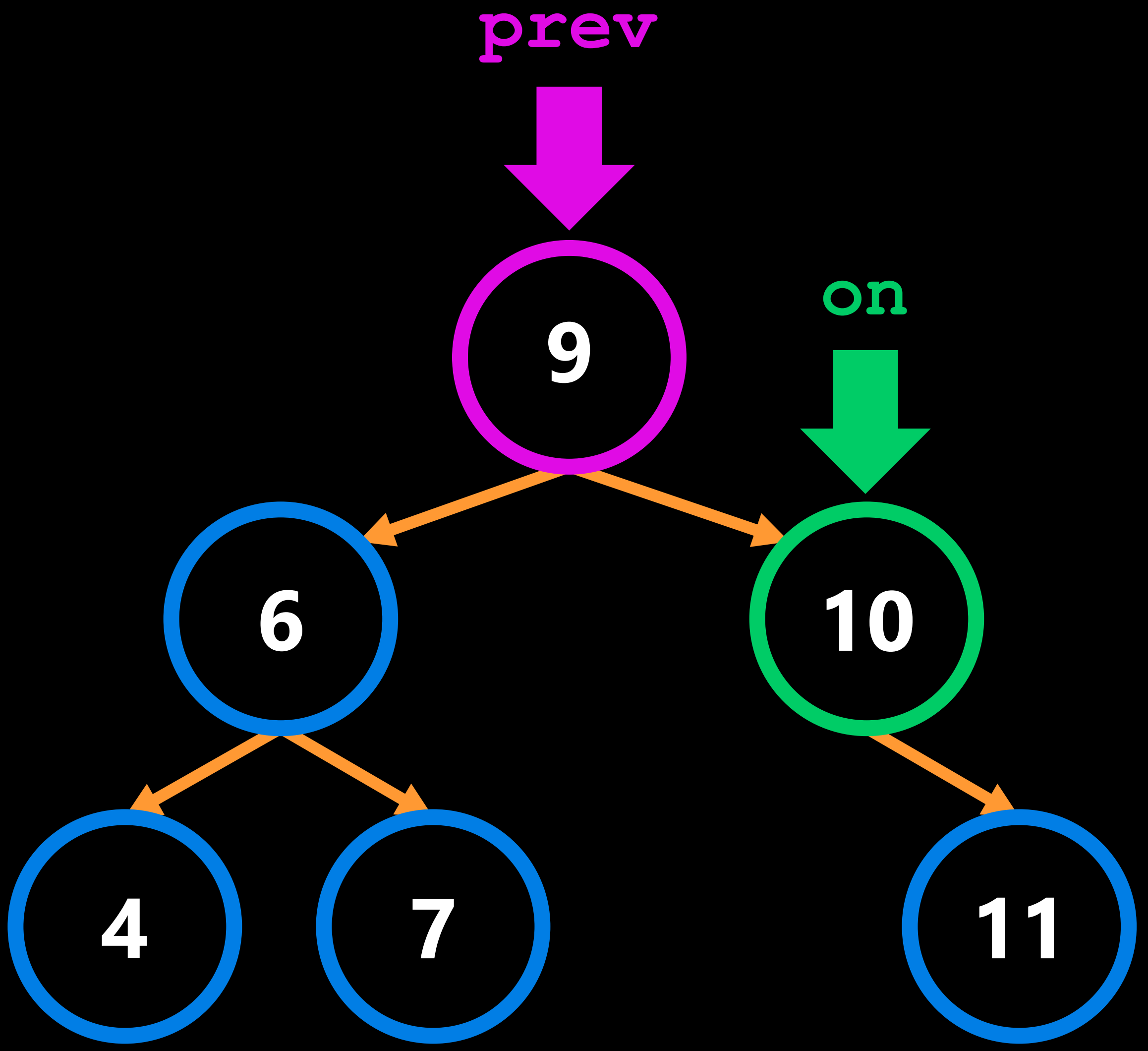
```python
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

True

False

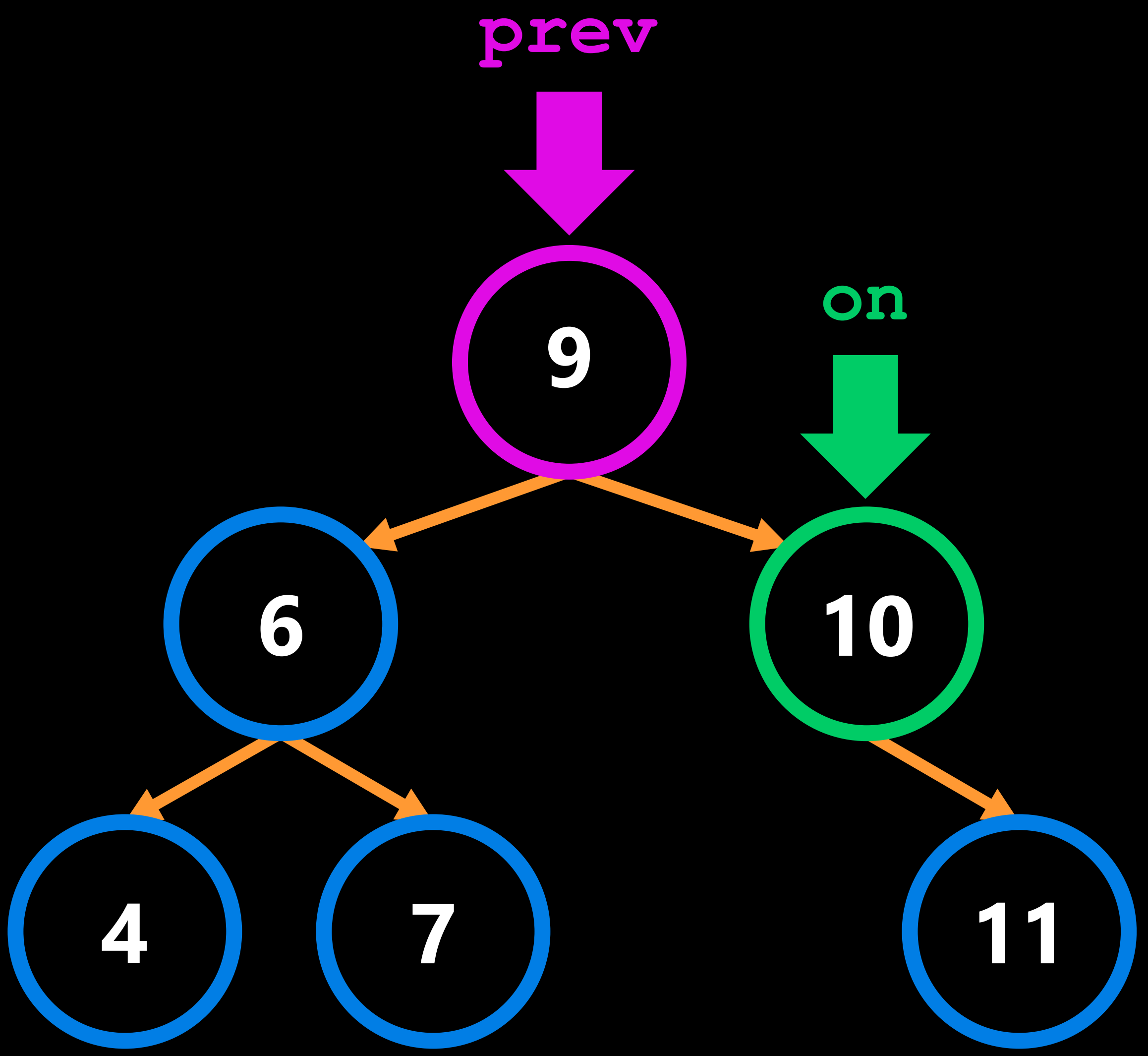
stack = [ ]

**Set on to left node in stack.**



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:  ← True

            while on is not None:  ← False
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:  ← False
                return False

            prev = on
            on = on.right

        return True
```

**stack = [ ]**



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

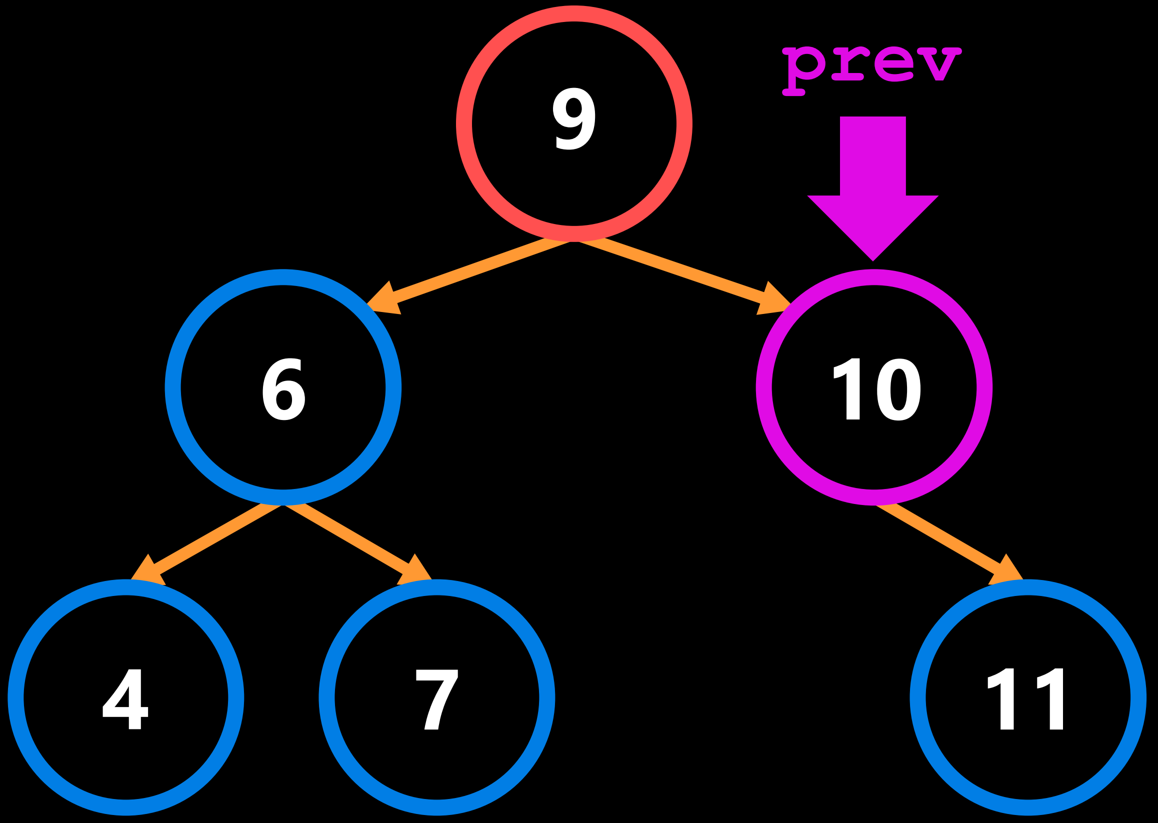
**stack = []**

True

False

False

**Set prev to on.**



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

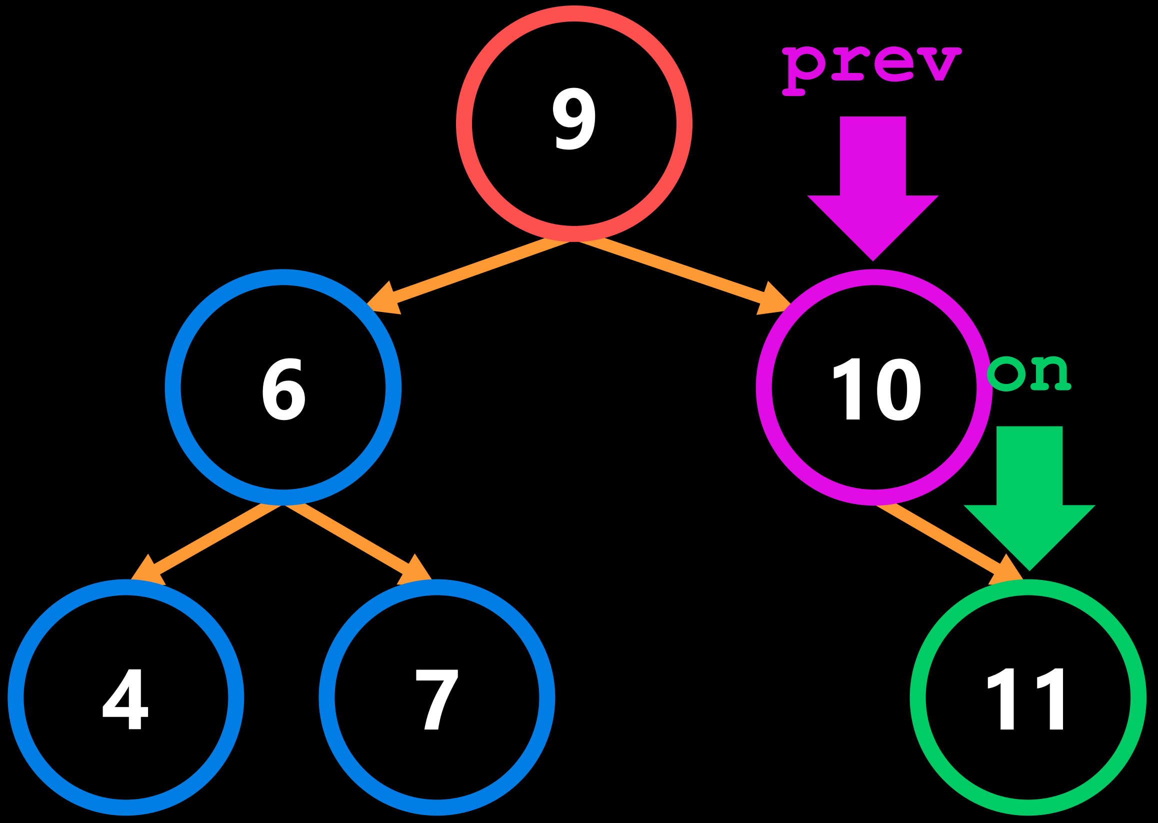
stack = [ ]

True

False

False

Move on to the right pointer.

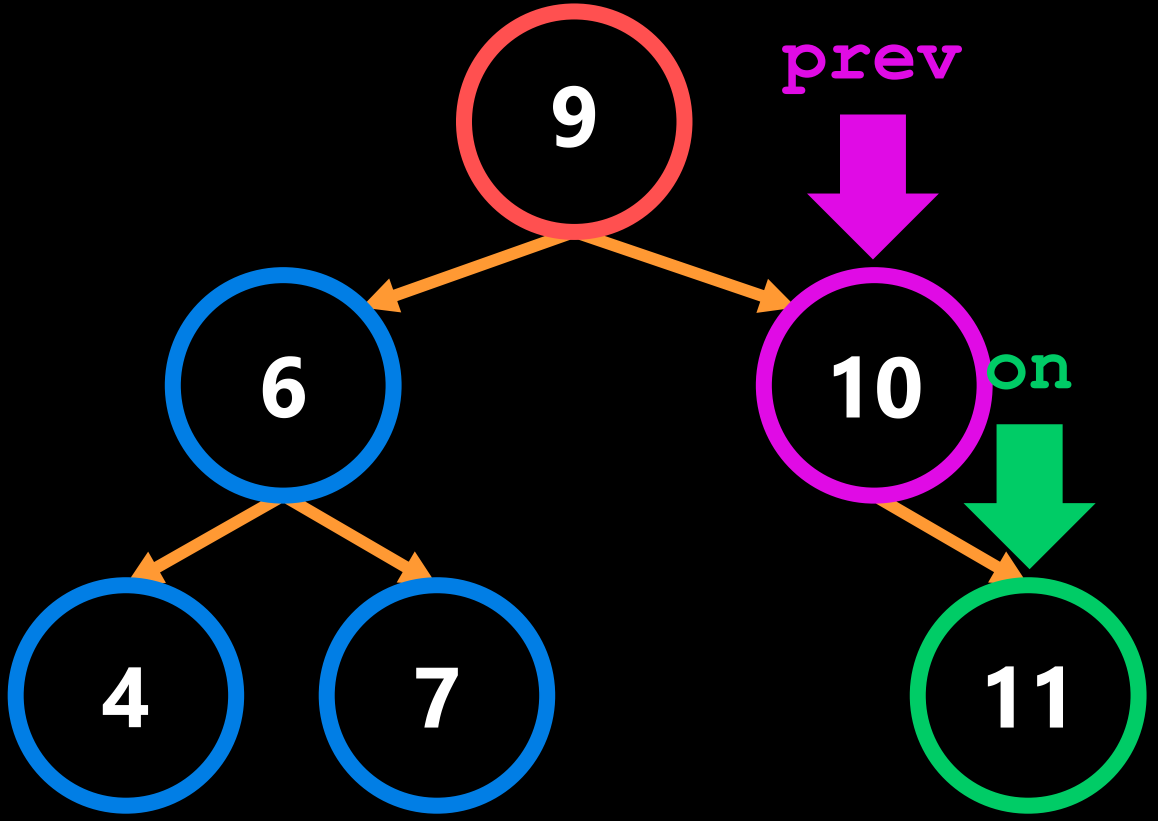


```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

True

True

stack = [ ]



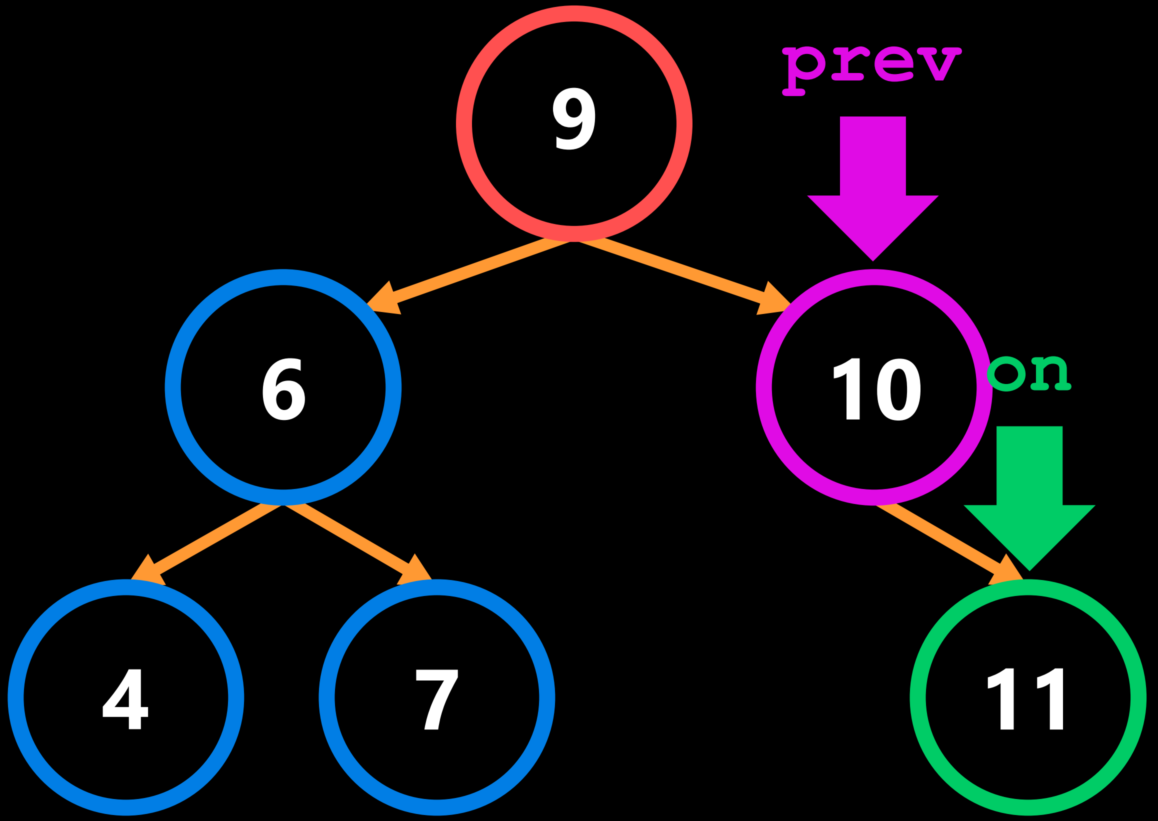
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

**stack = [11]**

True

True

**Add on to stack.**



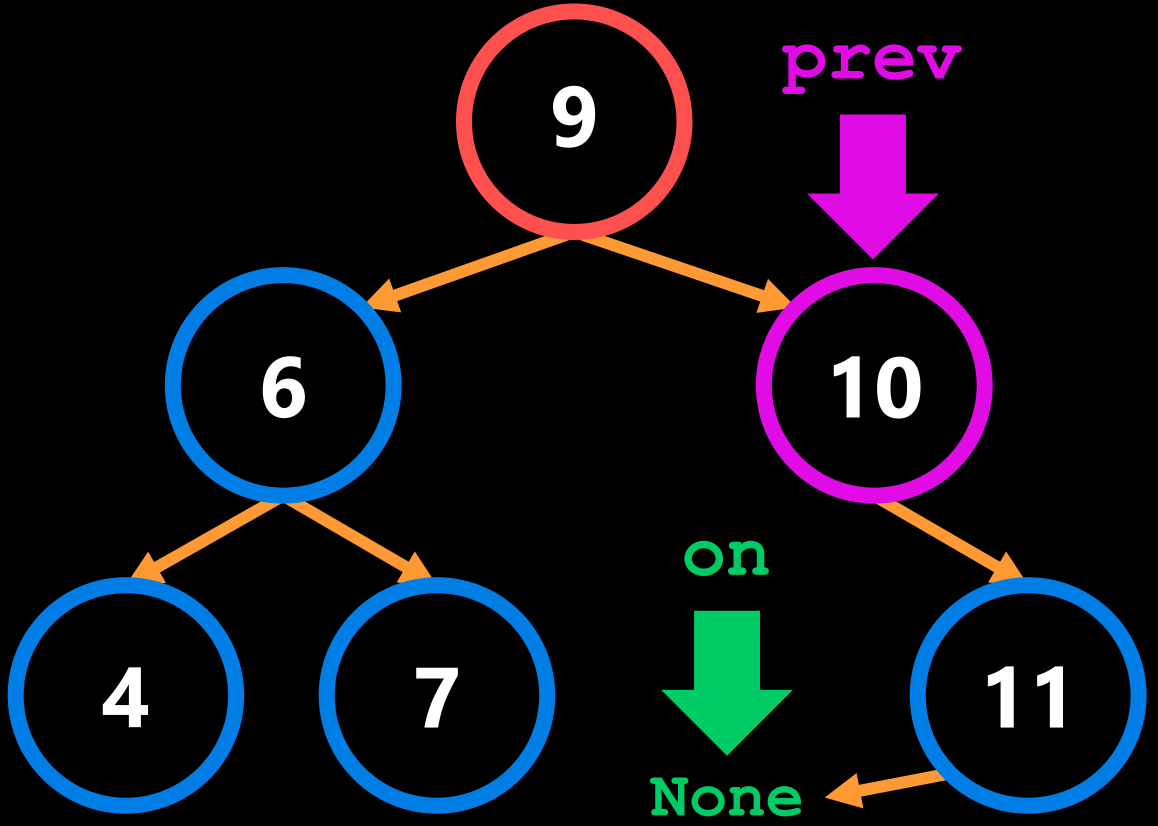
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

stack = [ 11 ]

True

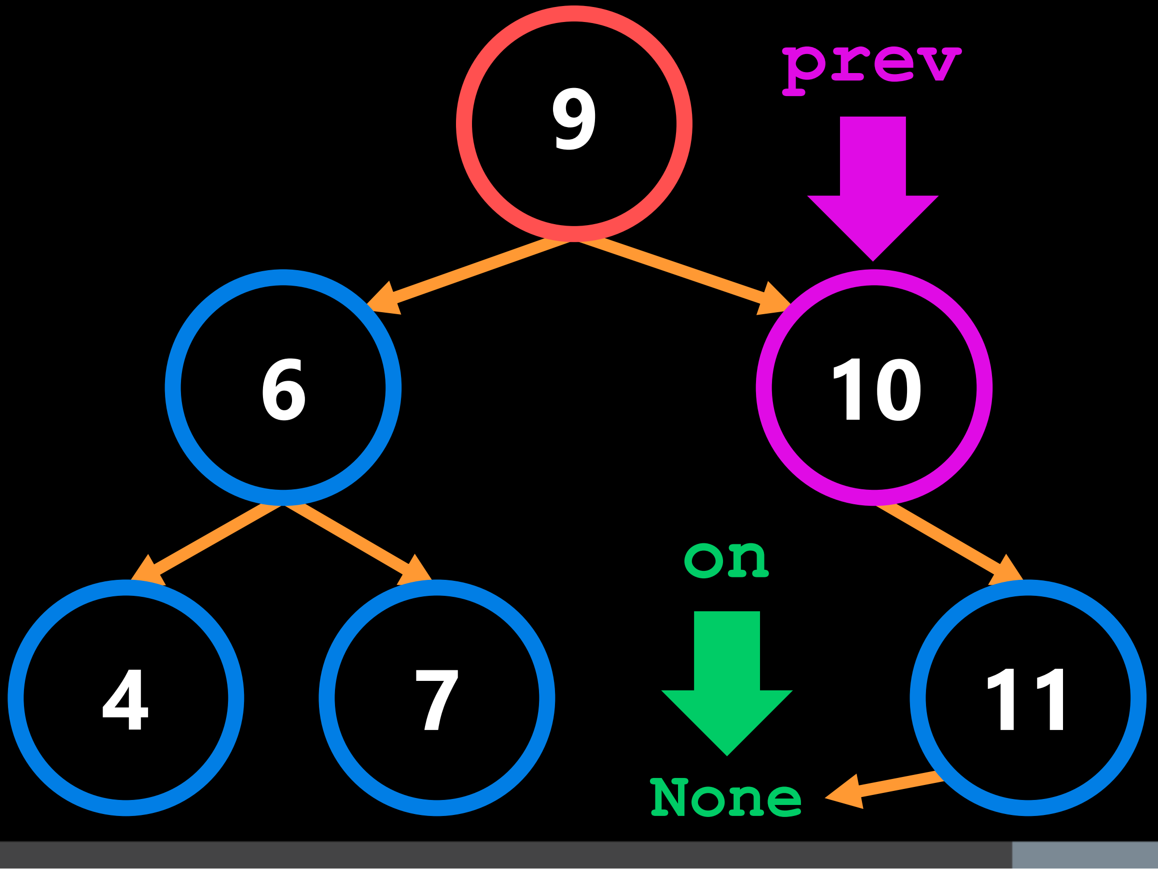
True

Move on to left node pointer.



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:      True

            while on is not None:      False
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

stack = [11]



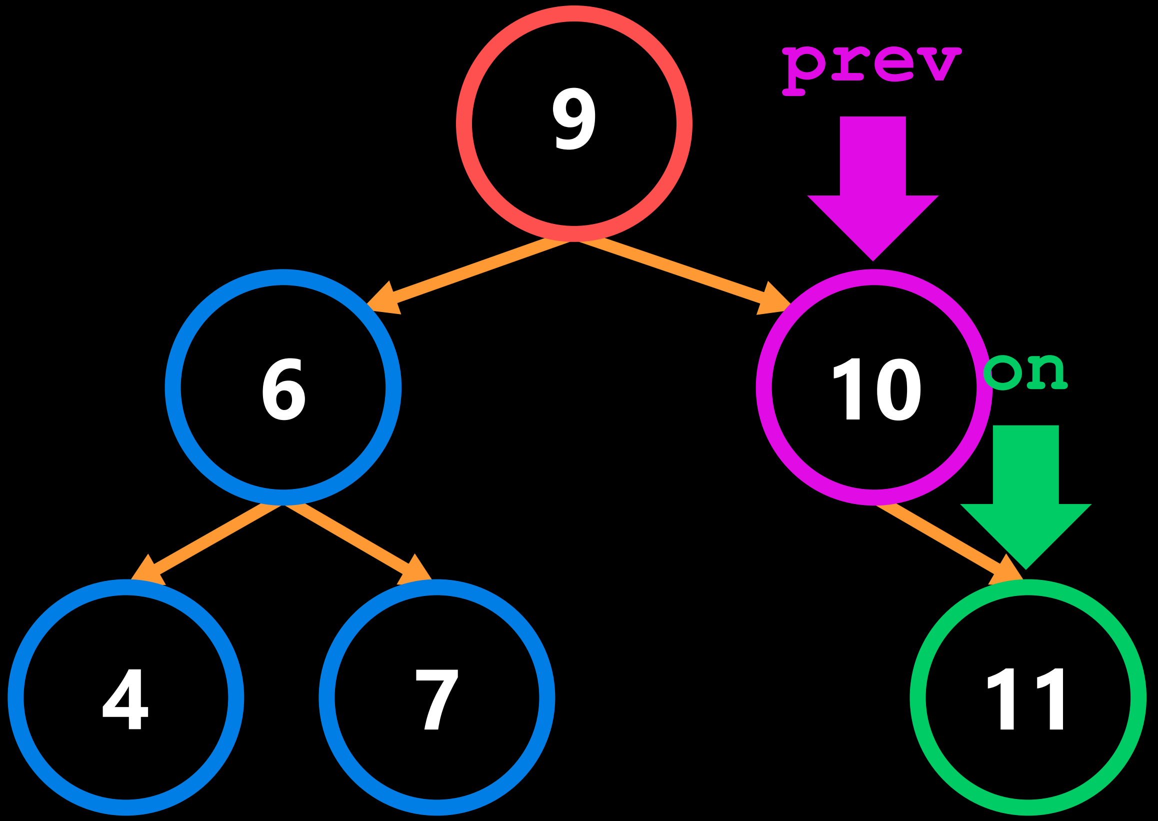
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

stack = [ ]

True

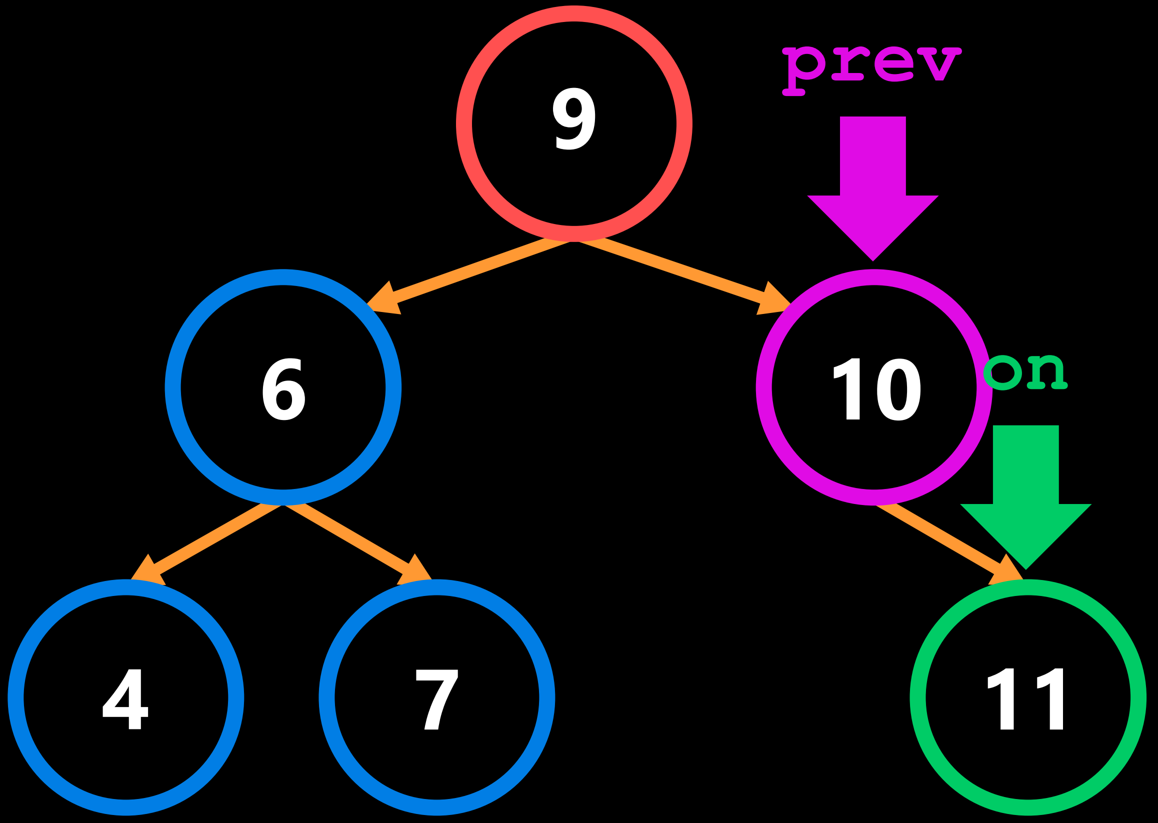
False

Set on to left node in stack.



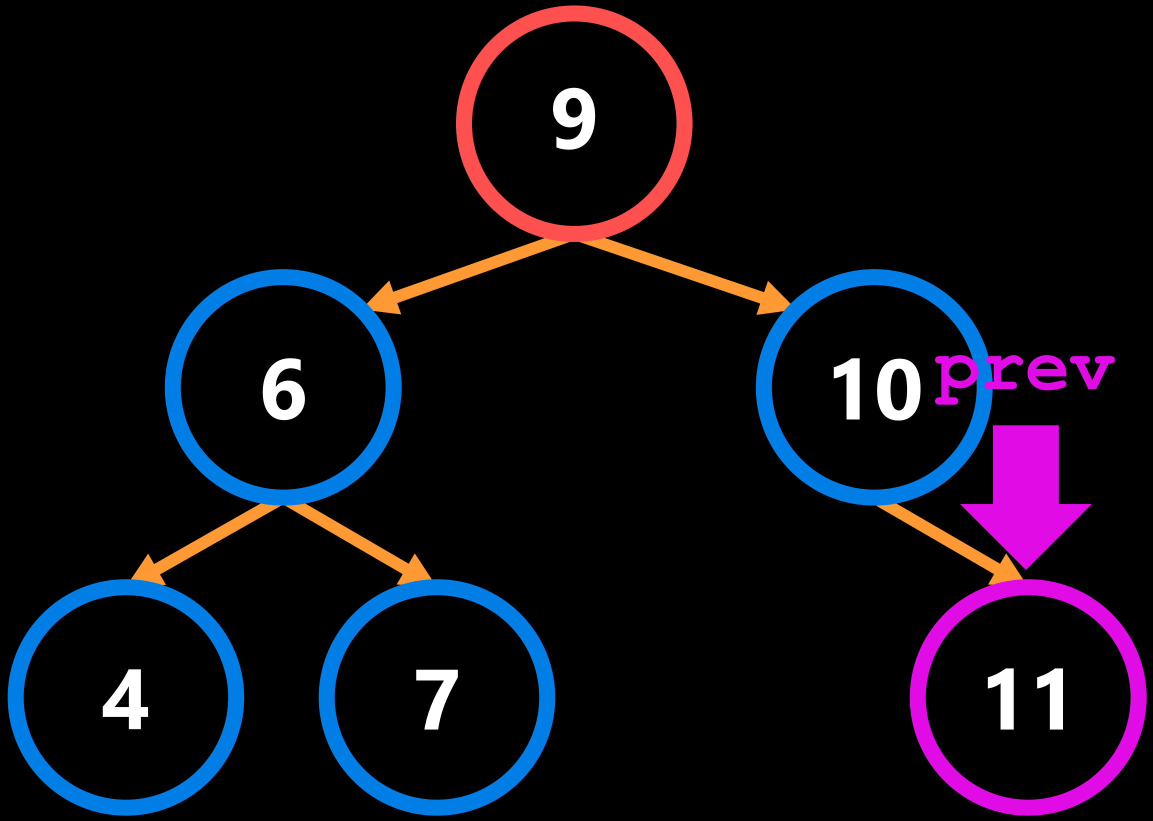
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:     True

            while on is not None:     False
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:     False
                return False

            prev = on
            on = on.right

        return True
```

**stack = [ ]**



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:  ← True

            while on is not None:  ← False
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:  ← False
                return False

            prev = on  ← Set prev to on.
            on = on.right

        return True
```

stack = [ ]



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

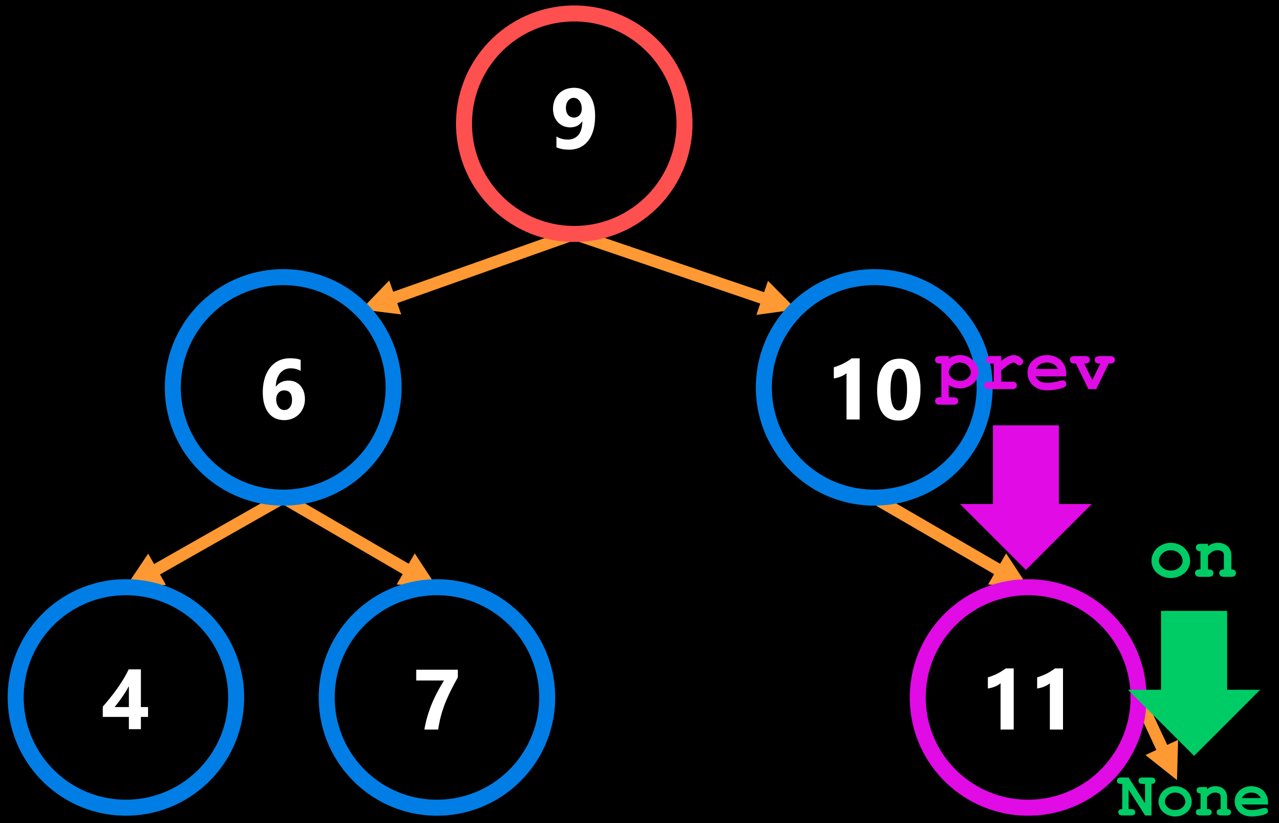
**stack = [ ]**

True

False

False

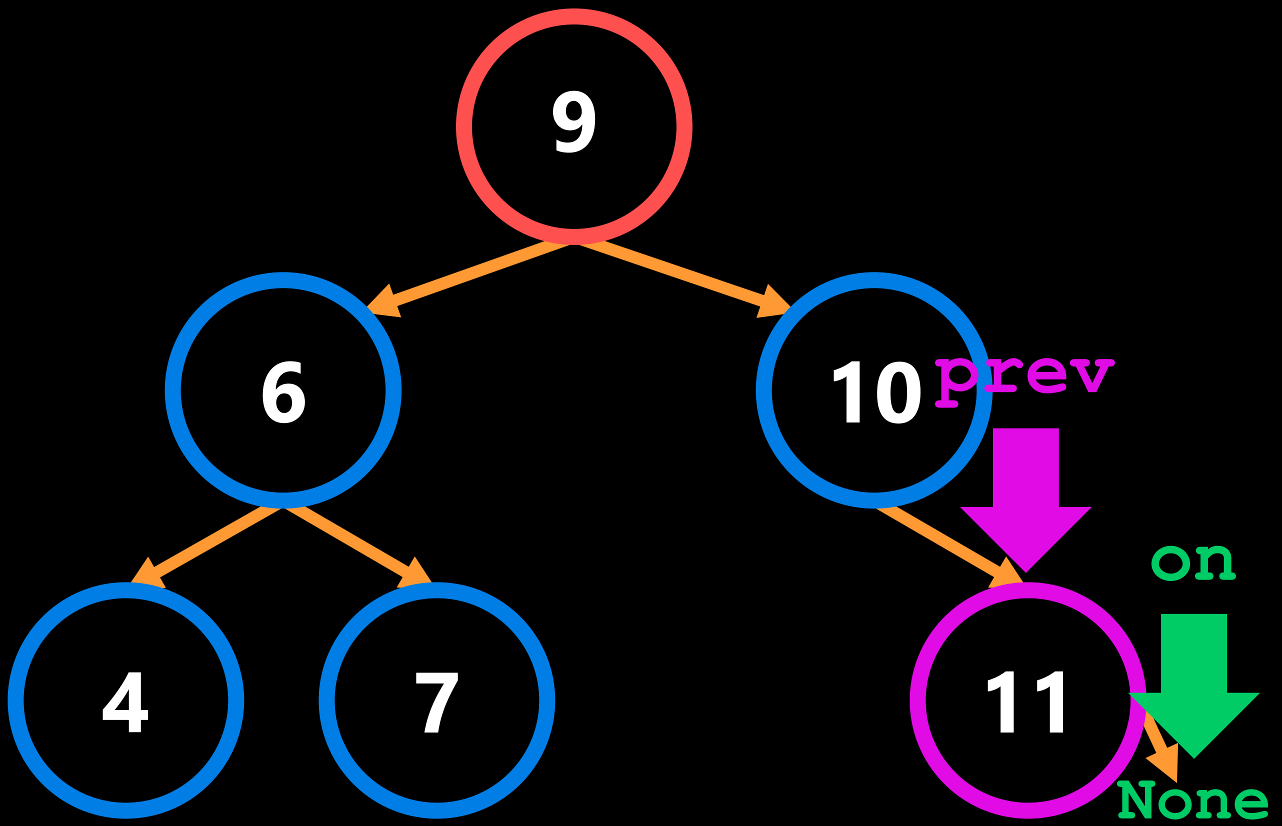
**Move on to the right pointer.**



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

False

stack = [ ]

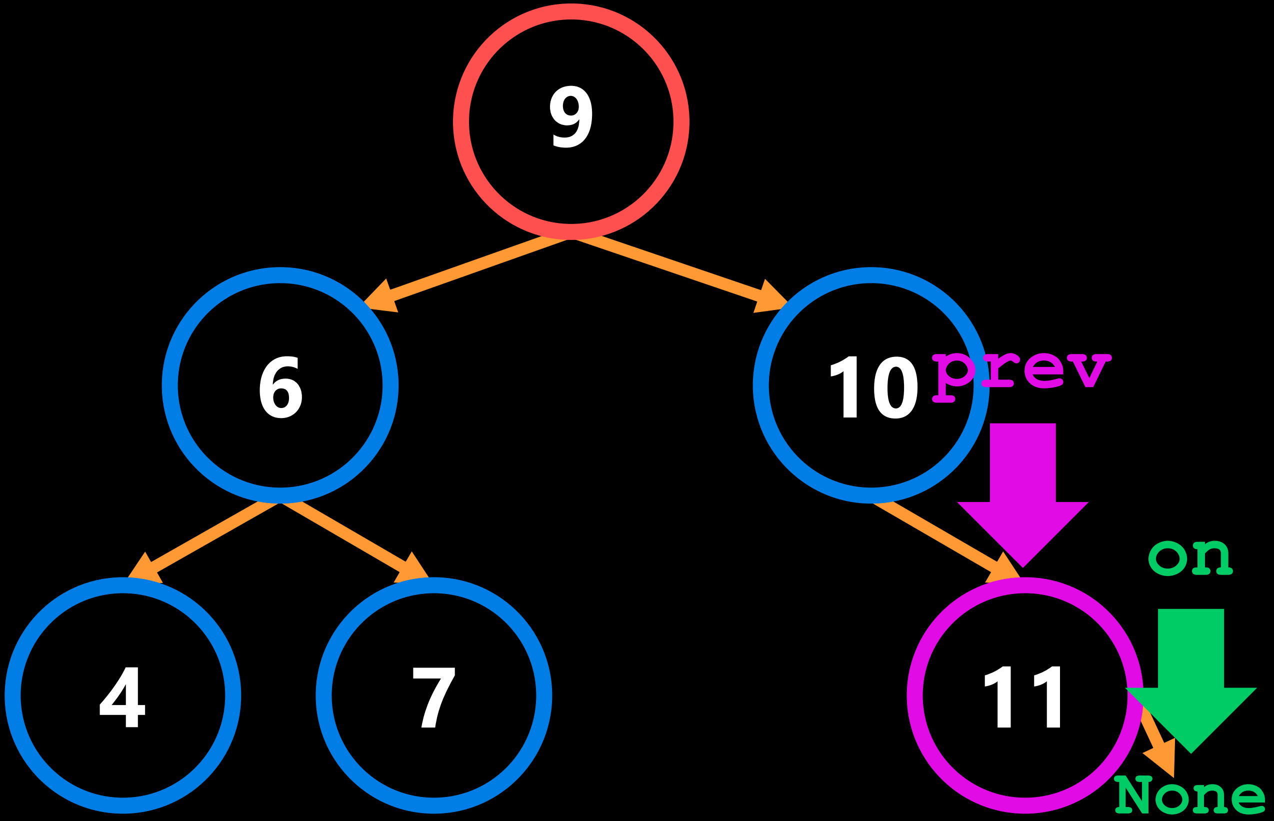


```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

Return True.

stack = []

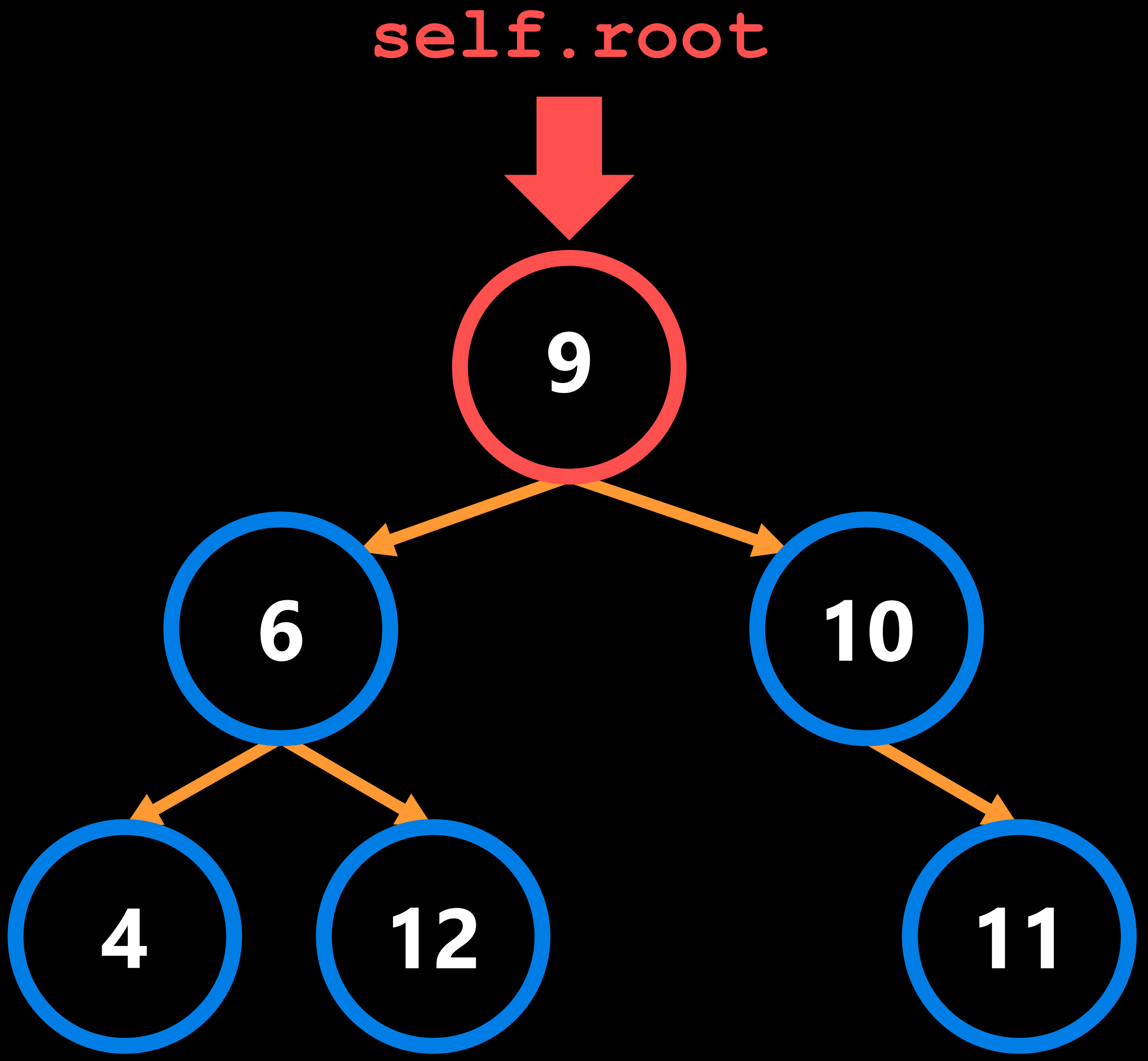
## This is a Valid Binary Search Tree!



# Let's try with an invalid binary search tree.

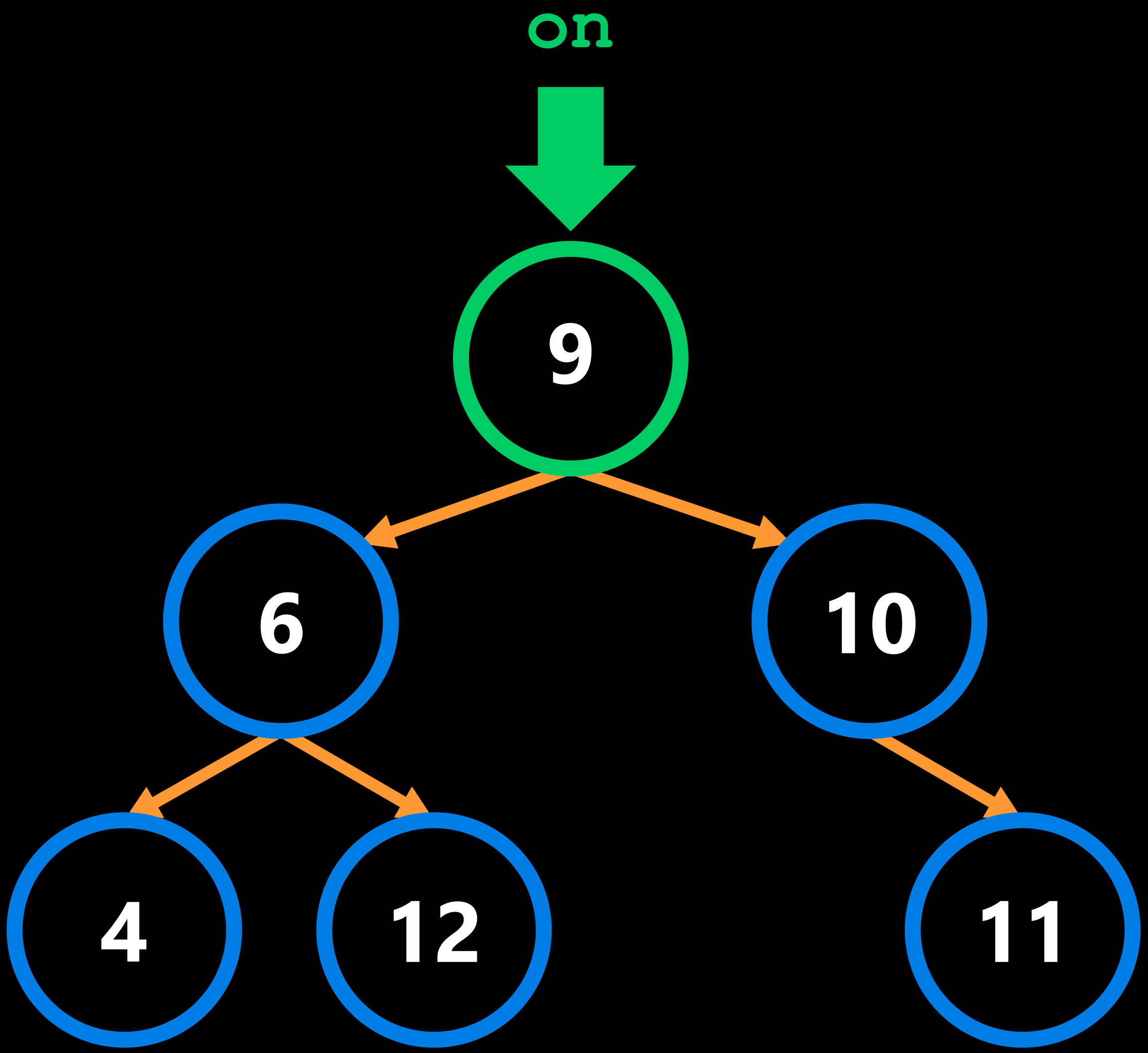
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

## This is an Invalid Tree



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

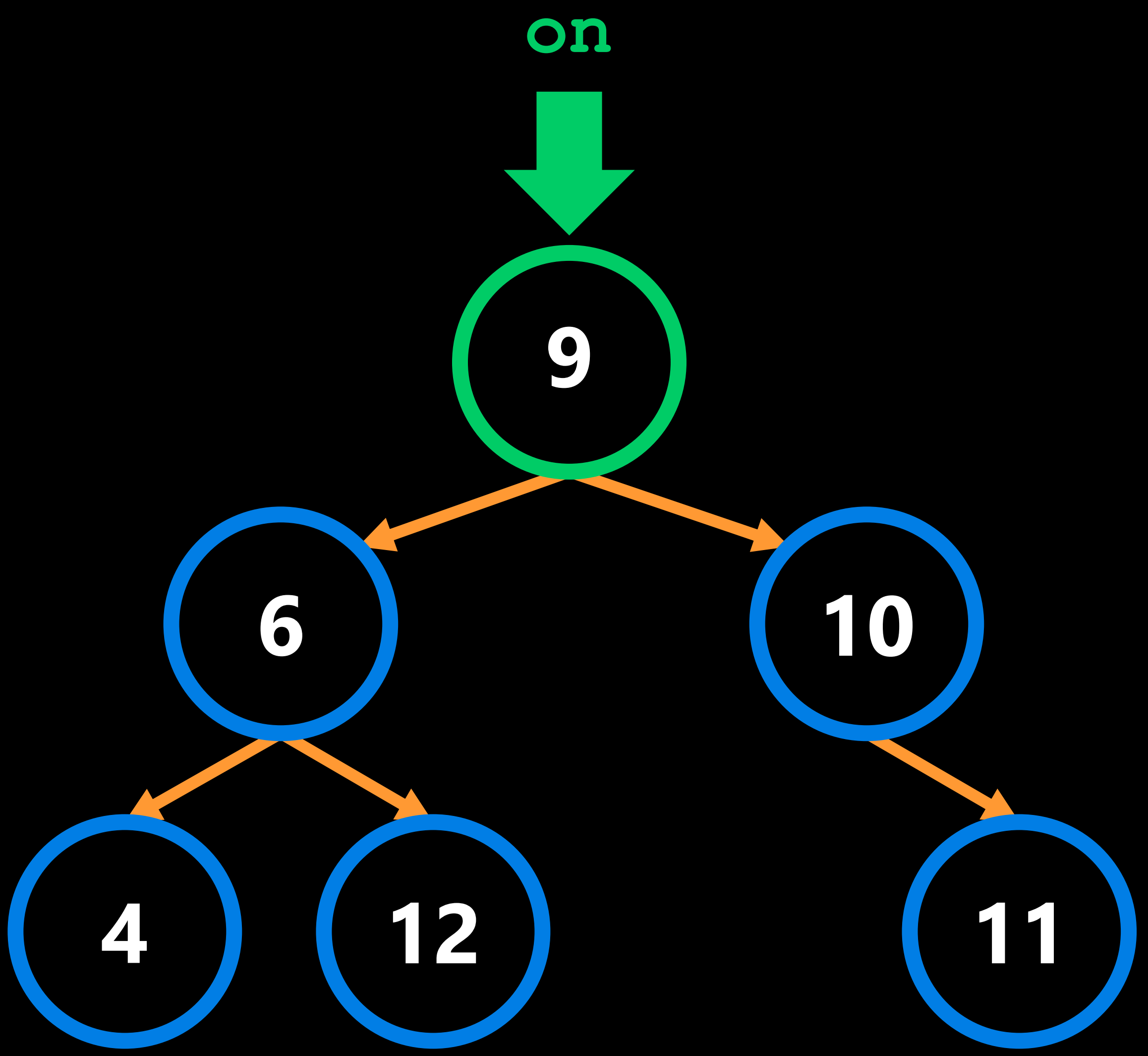
Set on position.



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

**Create stack list.**

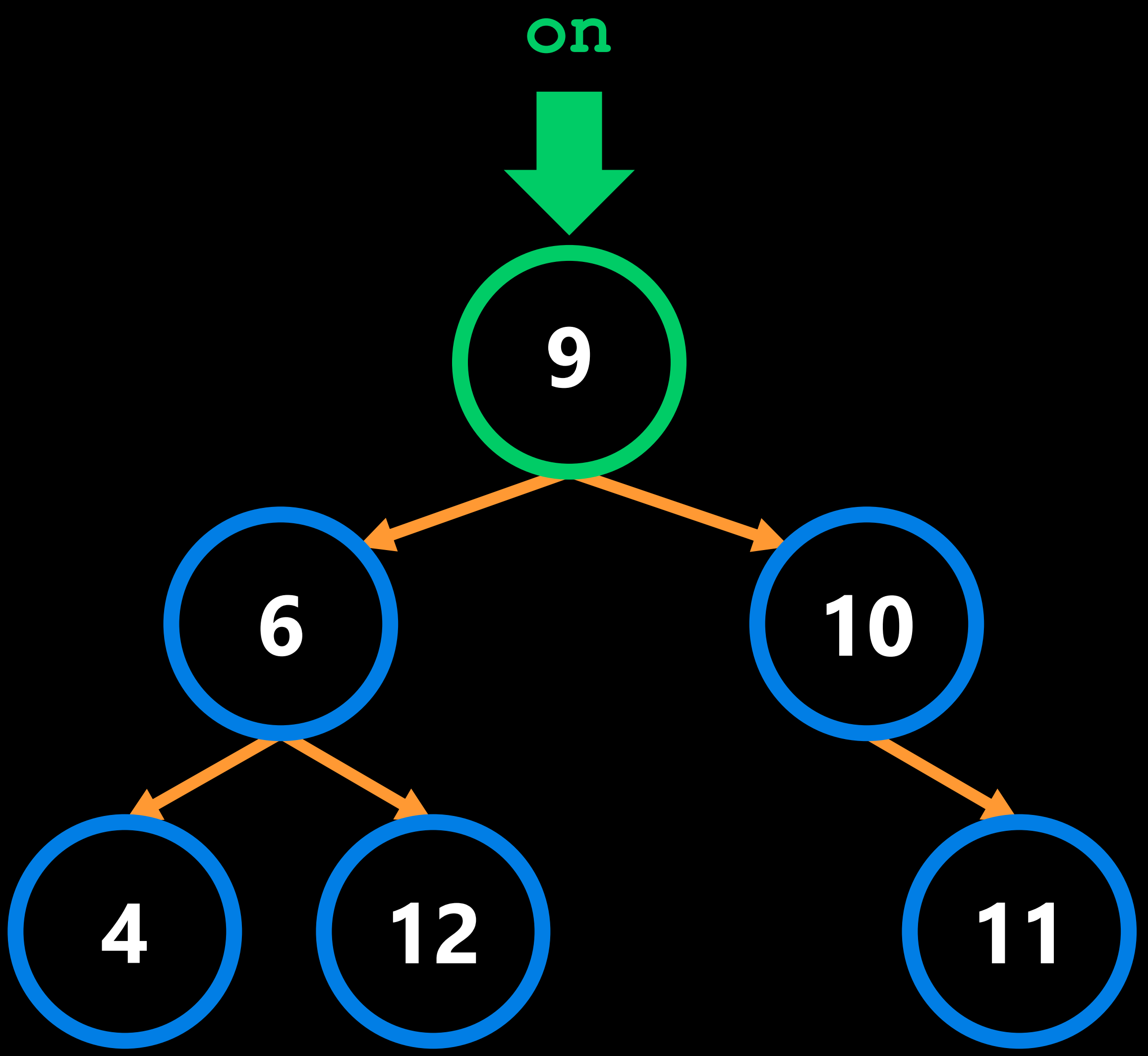
**stack = []**



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

**Initialize previous node.**

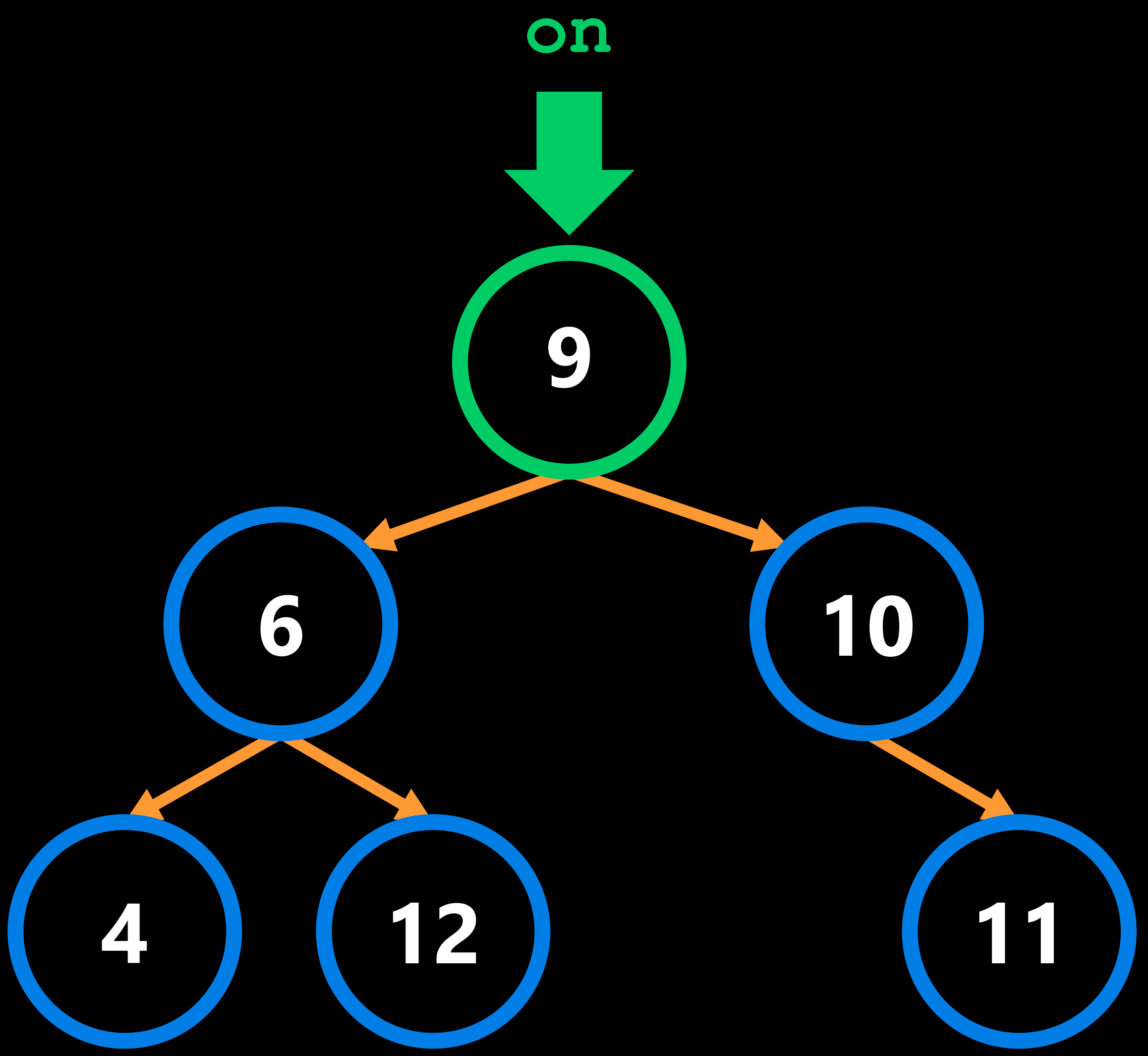
**stack = [ ]**



```python
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

True

stack = [ ]

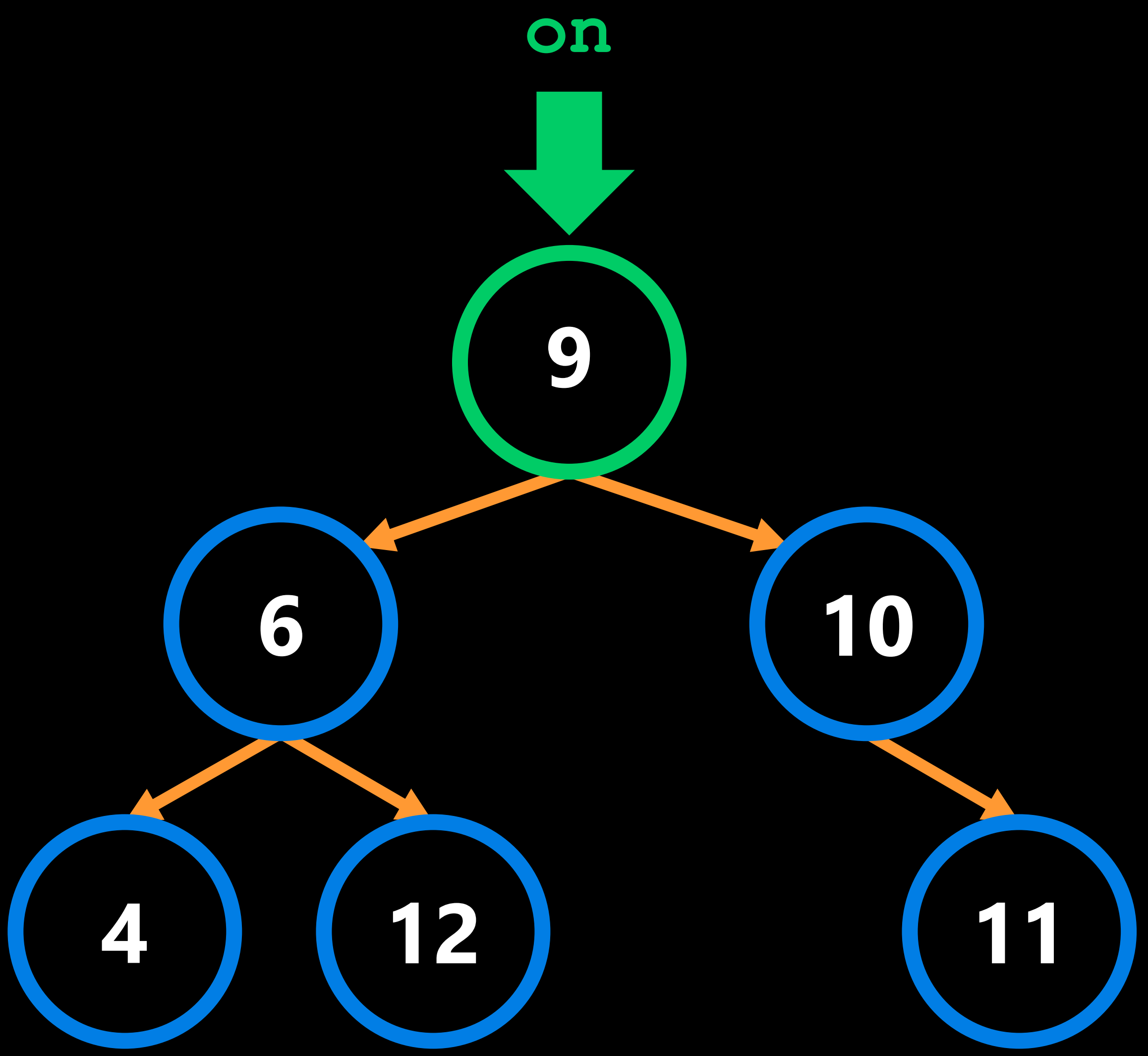


```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

True

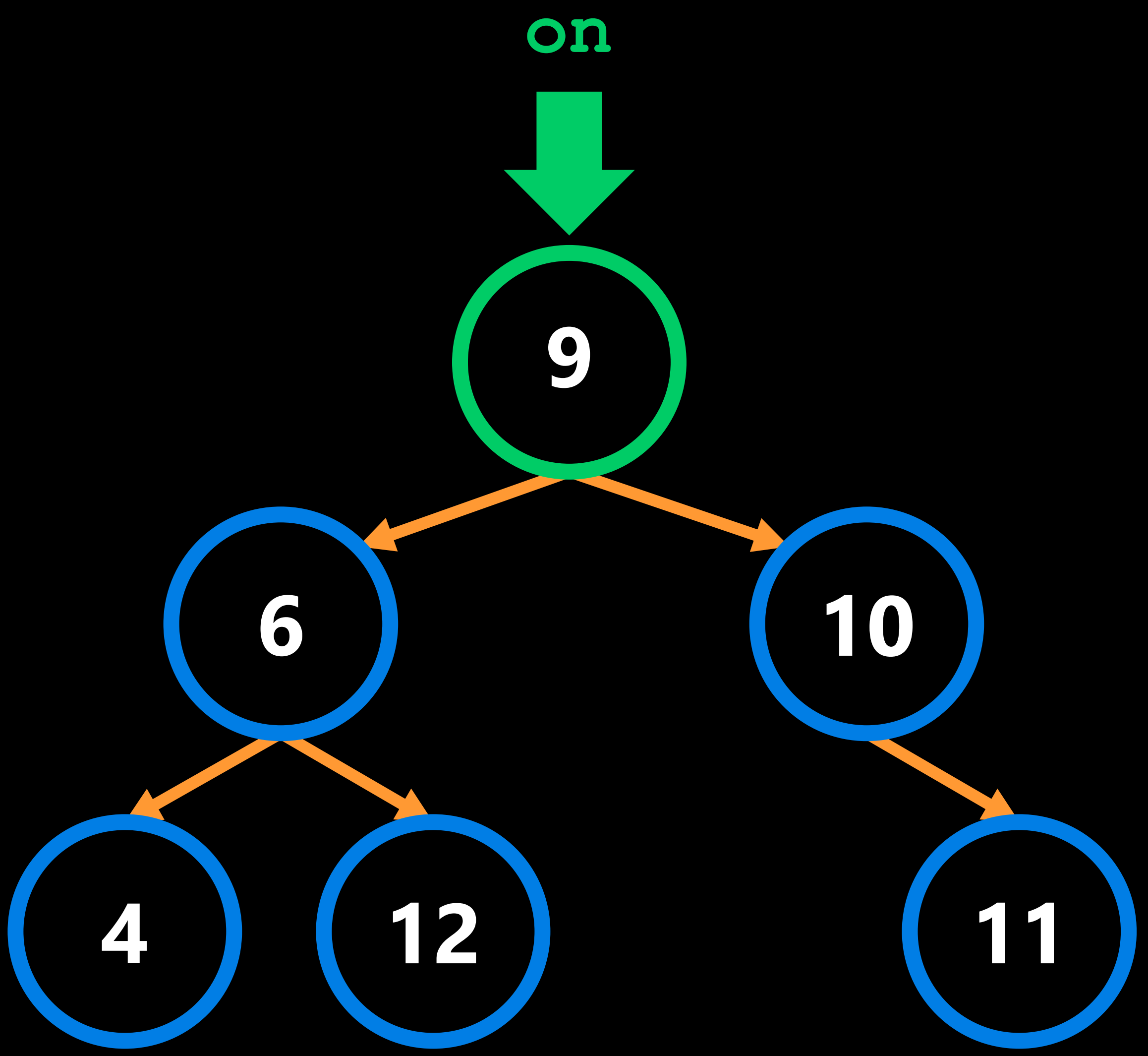
True

stack = [ ]



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:     True

            while on is not None:     True
                stack.append(on)      Add on to stack.
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

stack = [9]



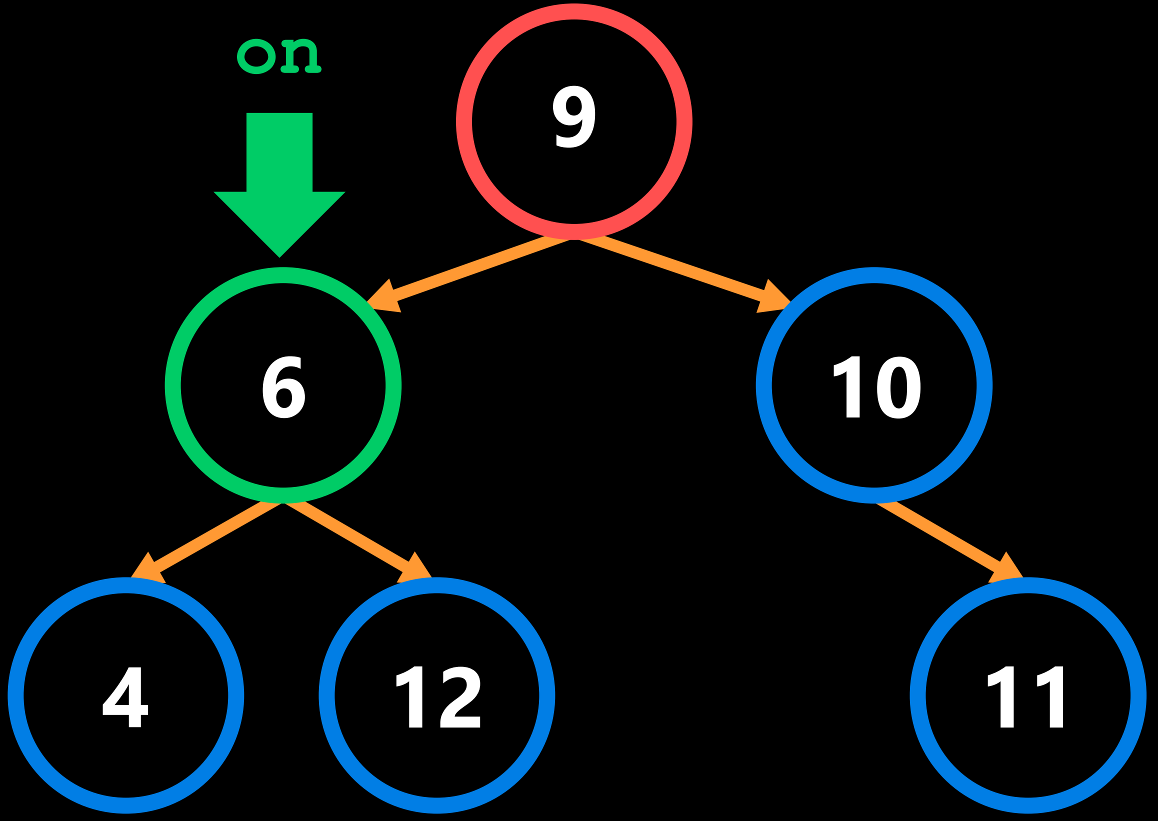
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

True

True

Move on to left node pointer.

stack = [9]



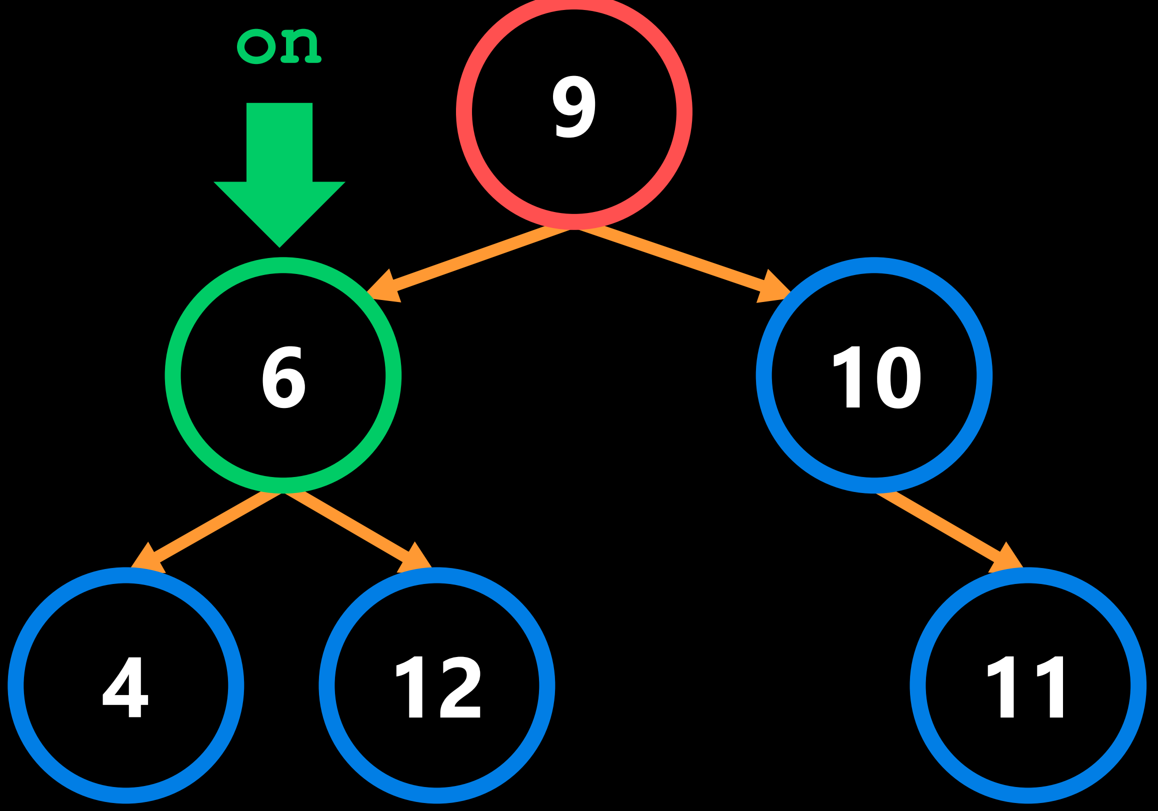
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

while len(stack) > 0 or on is not None: ← True

while on is not None: ← True

stack.append(on) ← Add on to stack.

stack = [9 6]



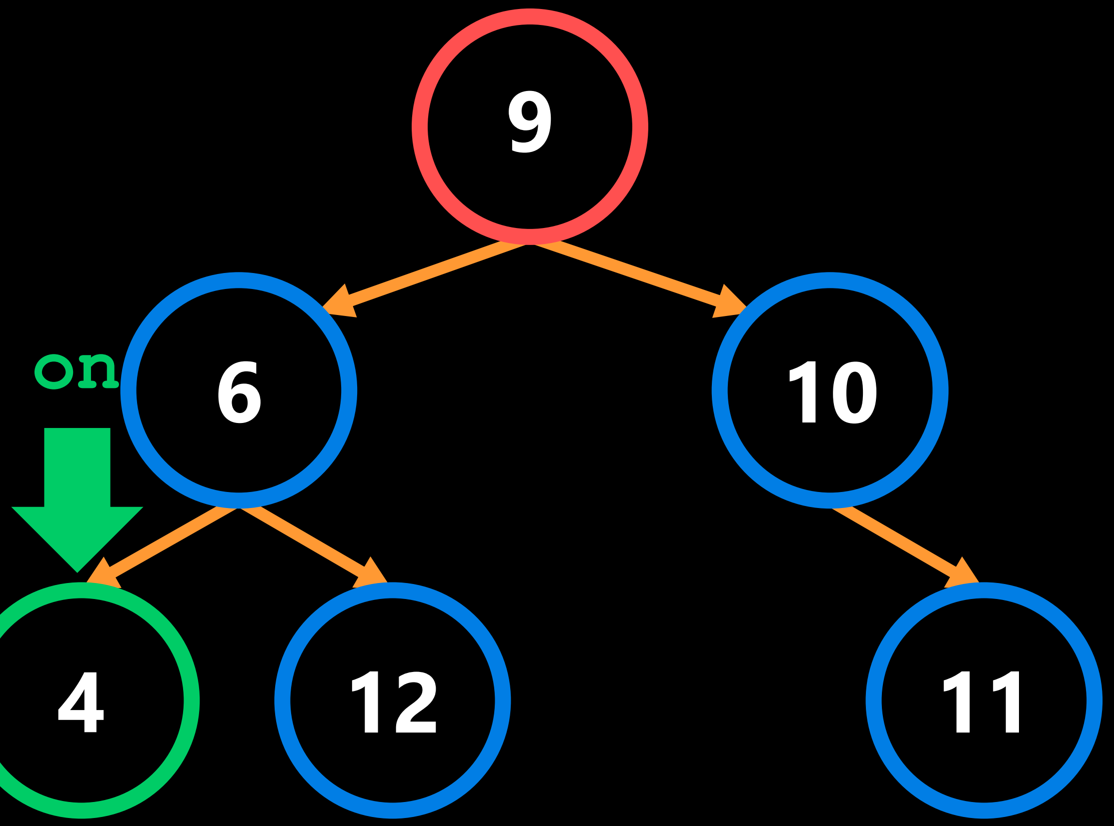
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

True

True

Move on to left node pointer.

stack = [ 9 6 ]



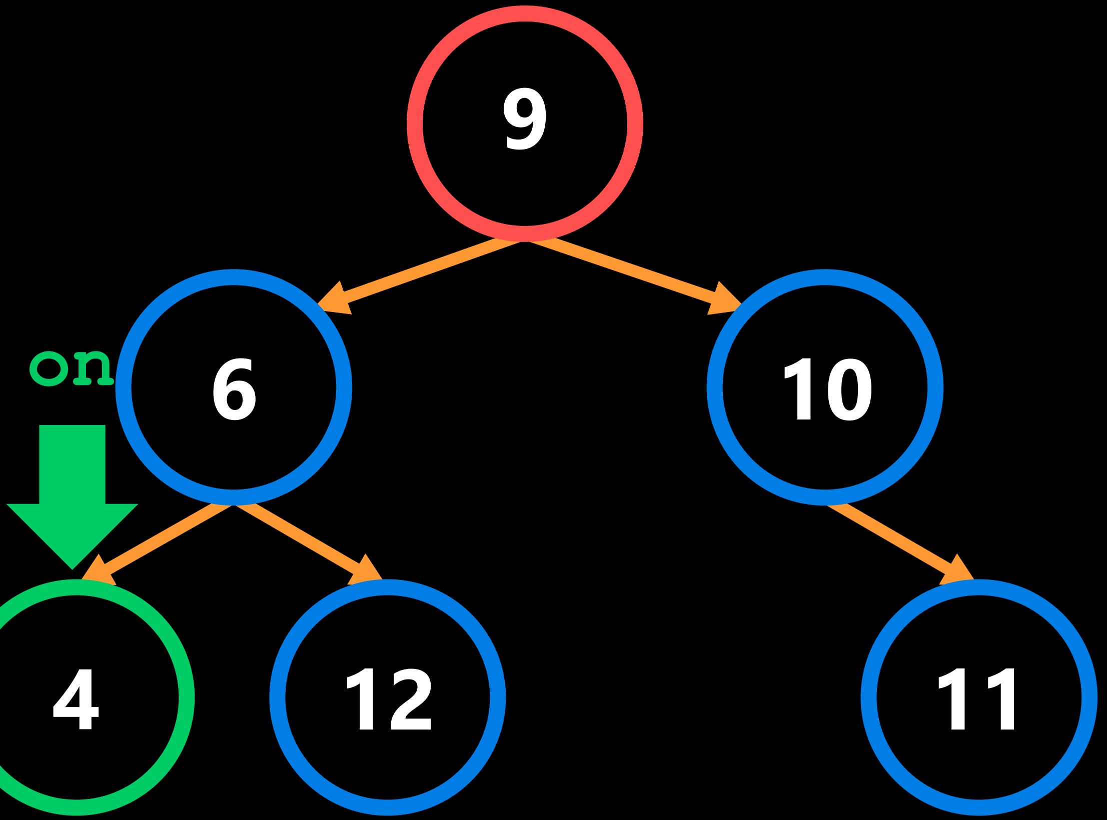
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

stack = [9 6 4]

while len(stack) > 0 or on is not None: ← True

while on is not None: ← True

stack.append(on) ← Add on to stack.



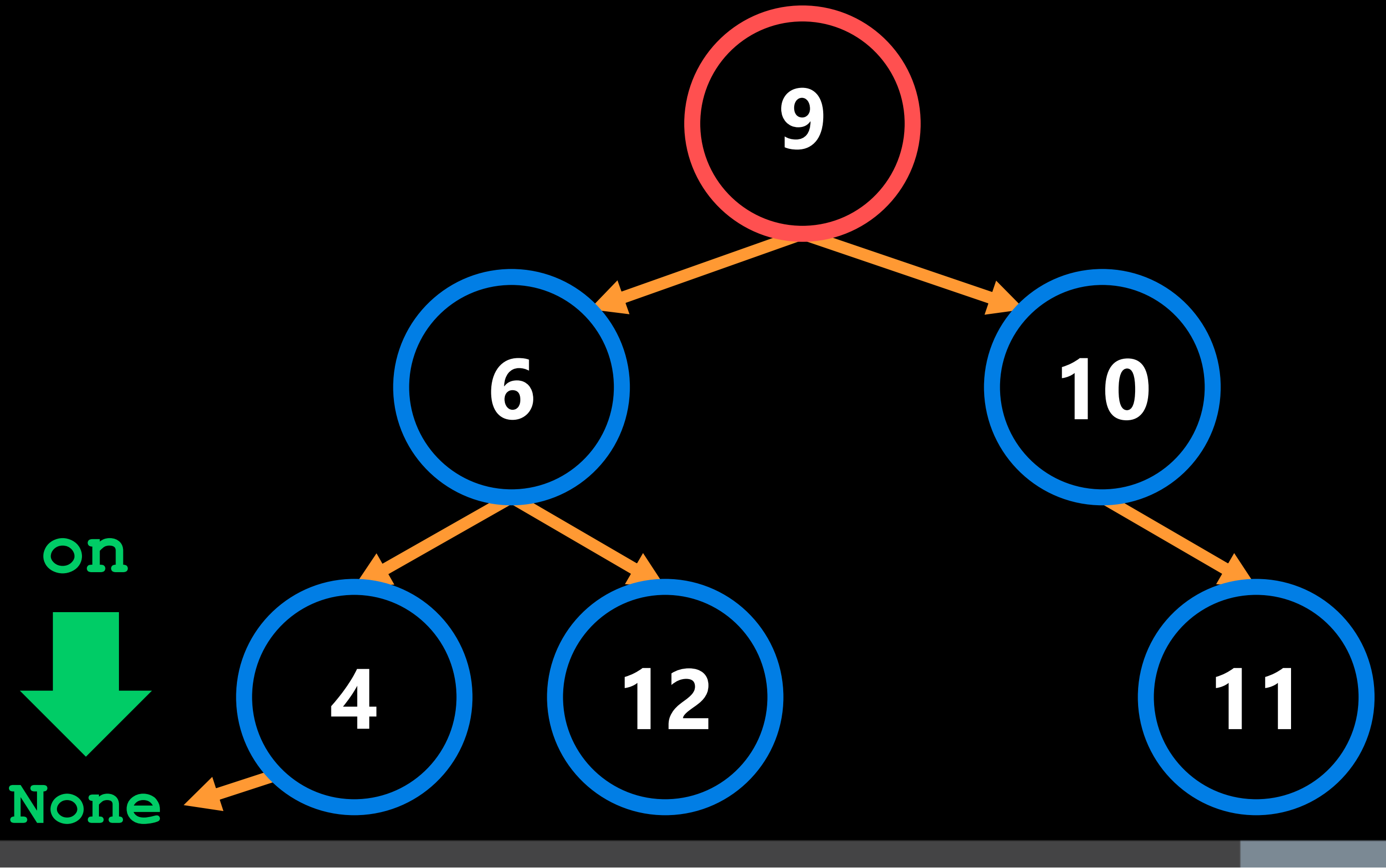
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

True

True

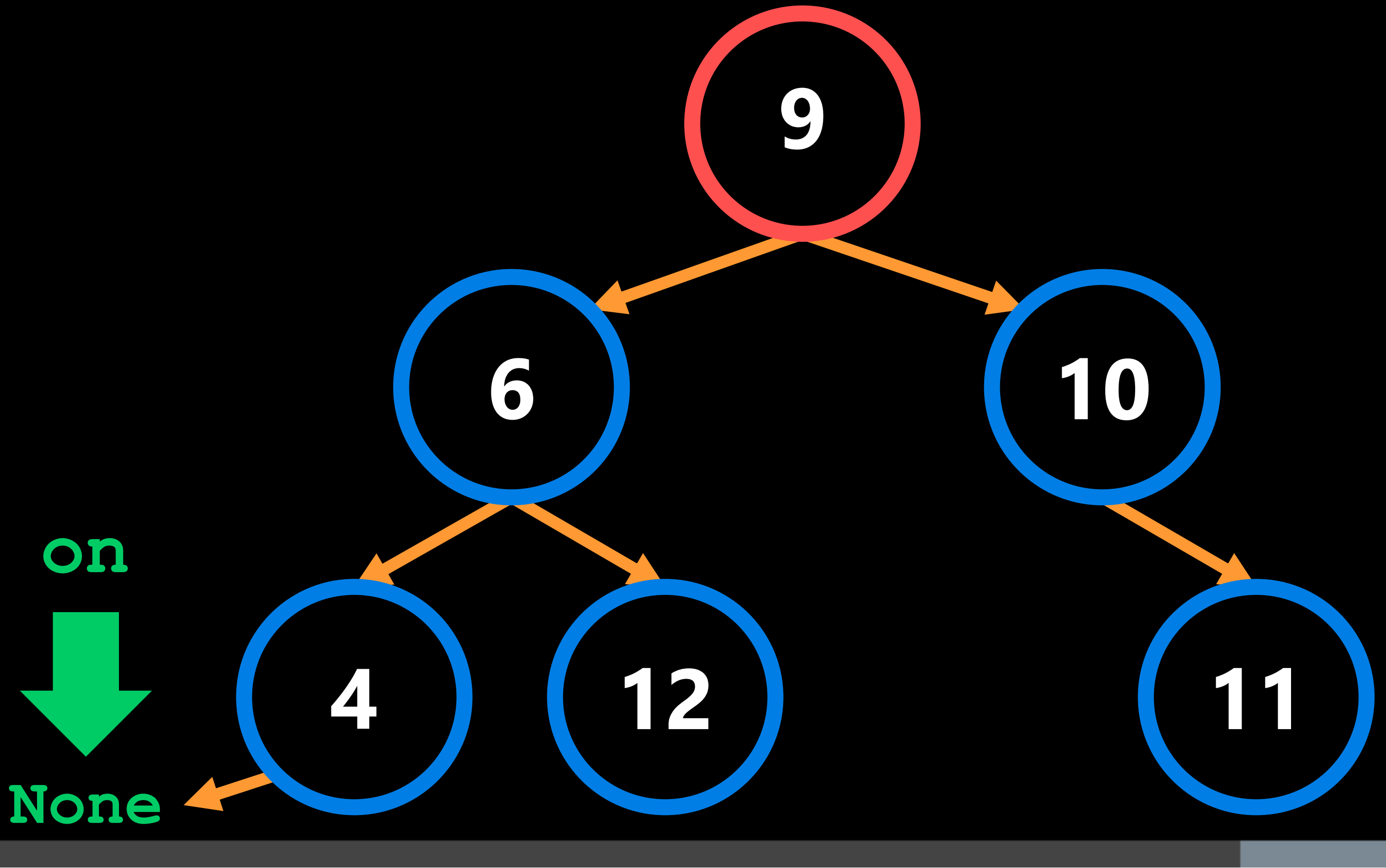
Move on to left node pointer.

stack = [ 9 6 4 ]



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:  ⬅ True

            while on is not None:  ⬅ False
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

stack = [9 6 4]



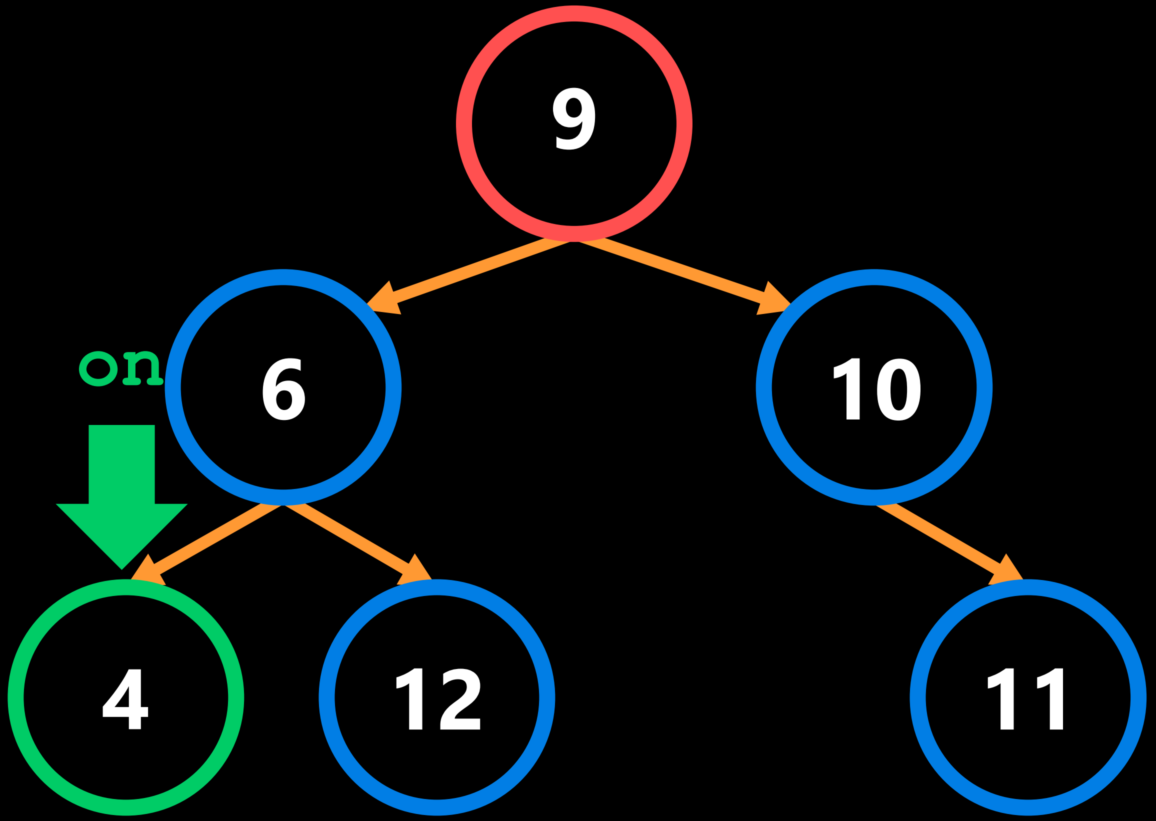
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

stack = [9 6]

True

False

Set on to left node in stack.



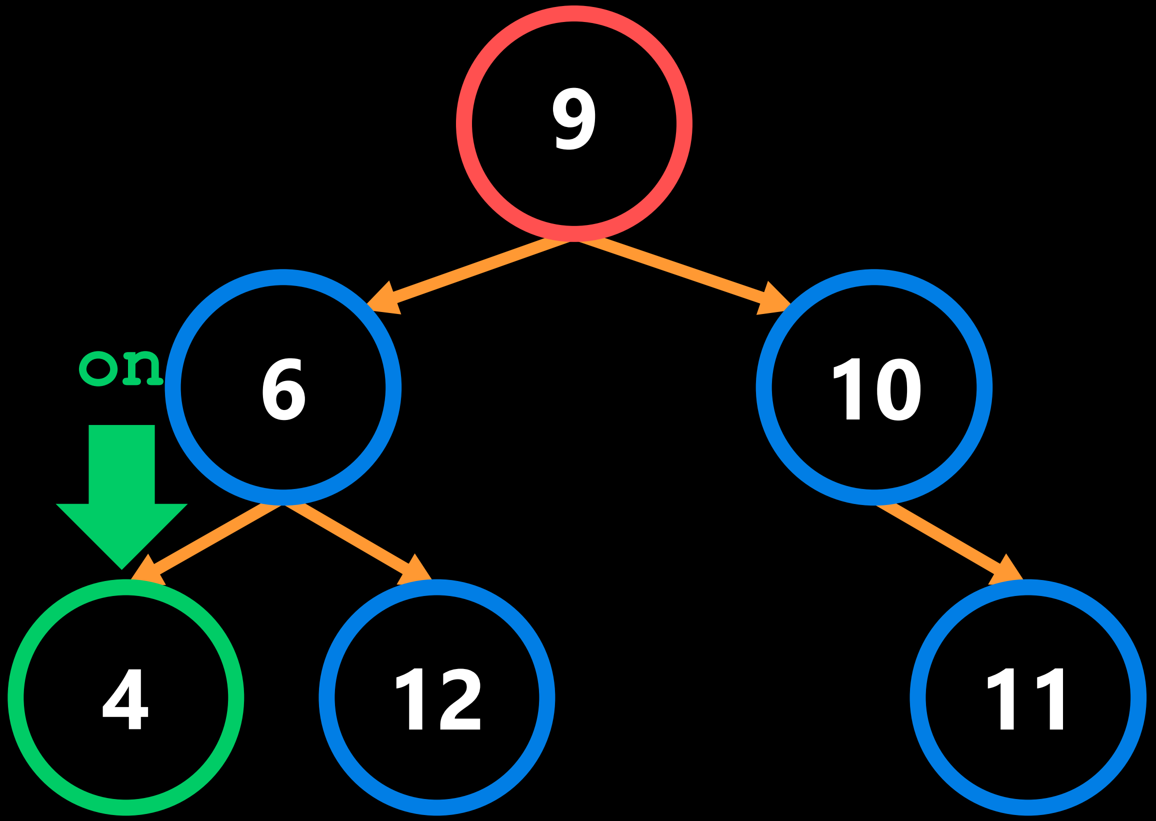
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

True

False

False

stack = [9 6]



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

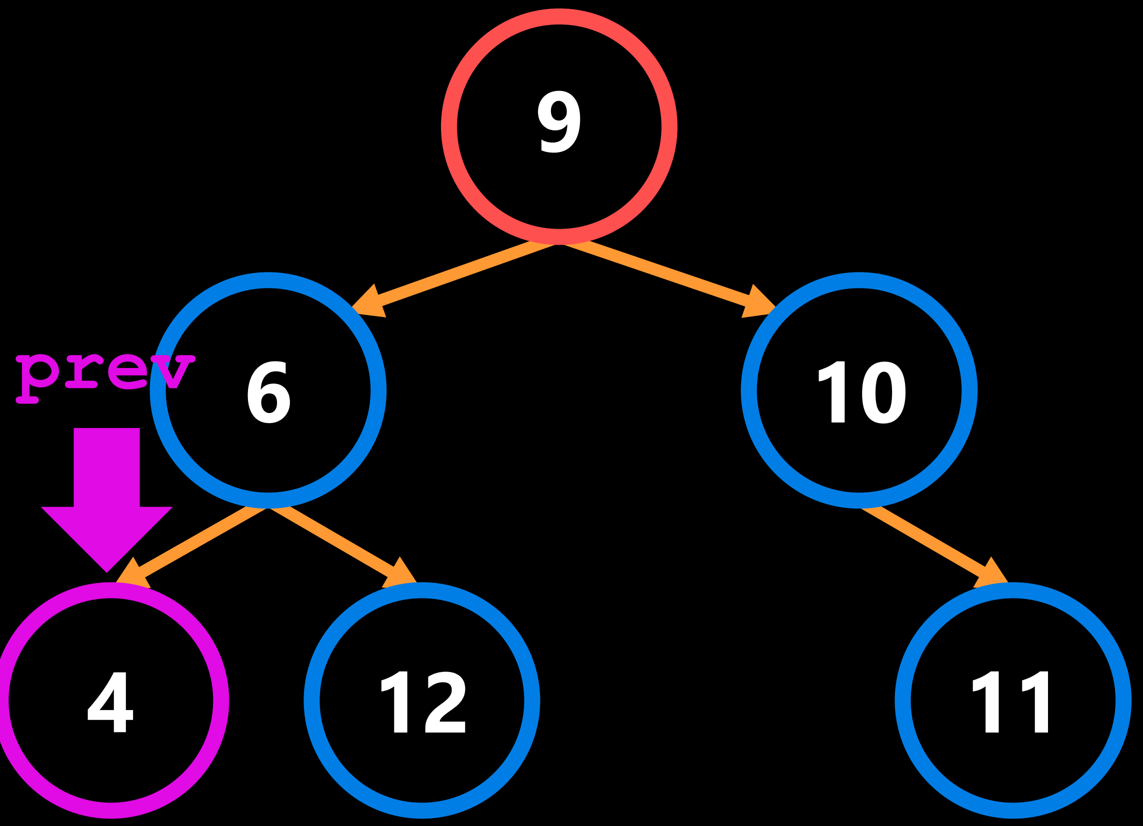
True

False

False

Set **prev** to **on.**

**stack = [ 9 6 ]**



```python
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

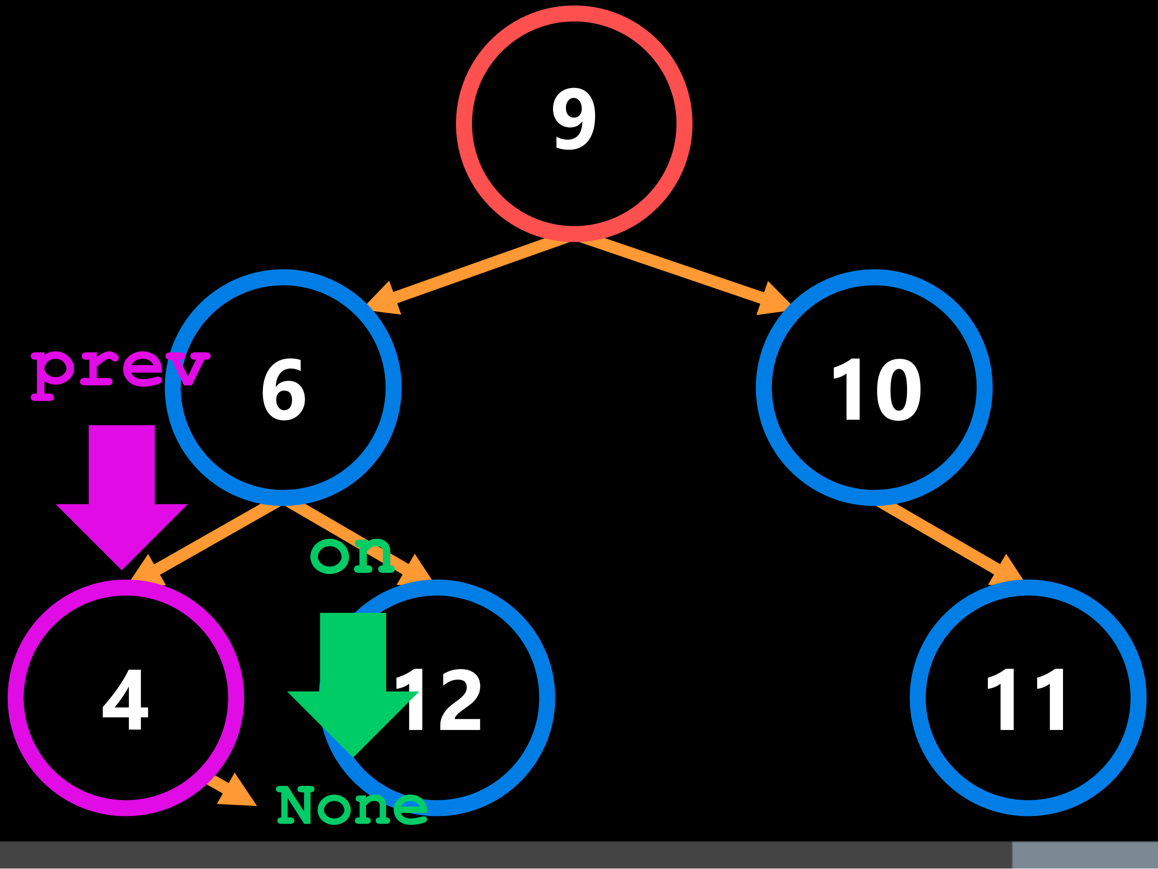
**stack = [9 6]**

True

False

False

**Move on to the right pointer.**



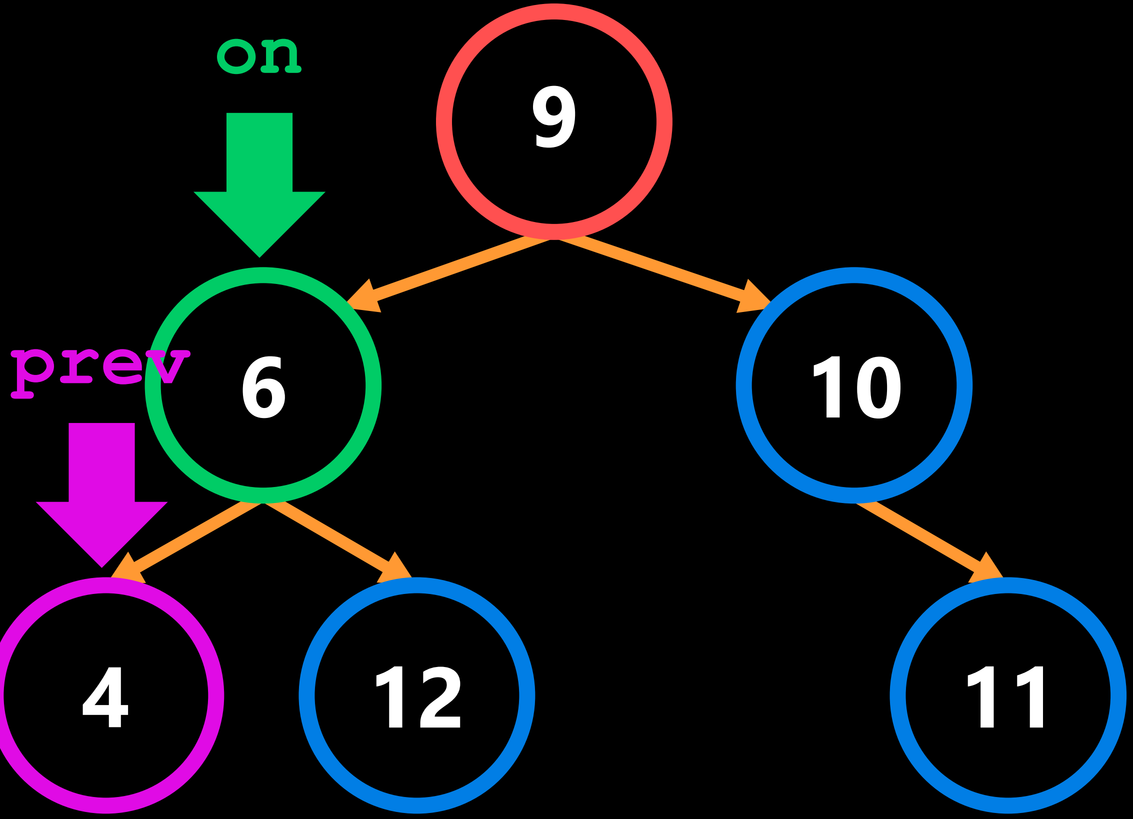
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

stack = [9]

True

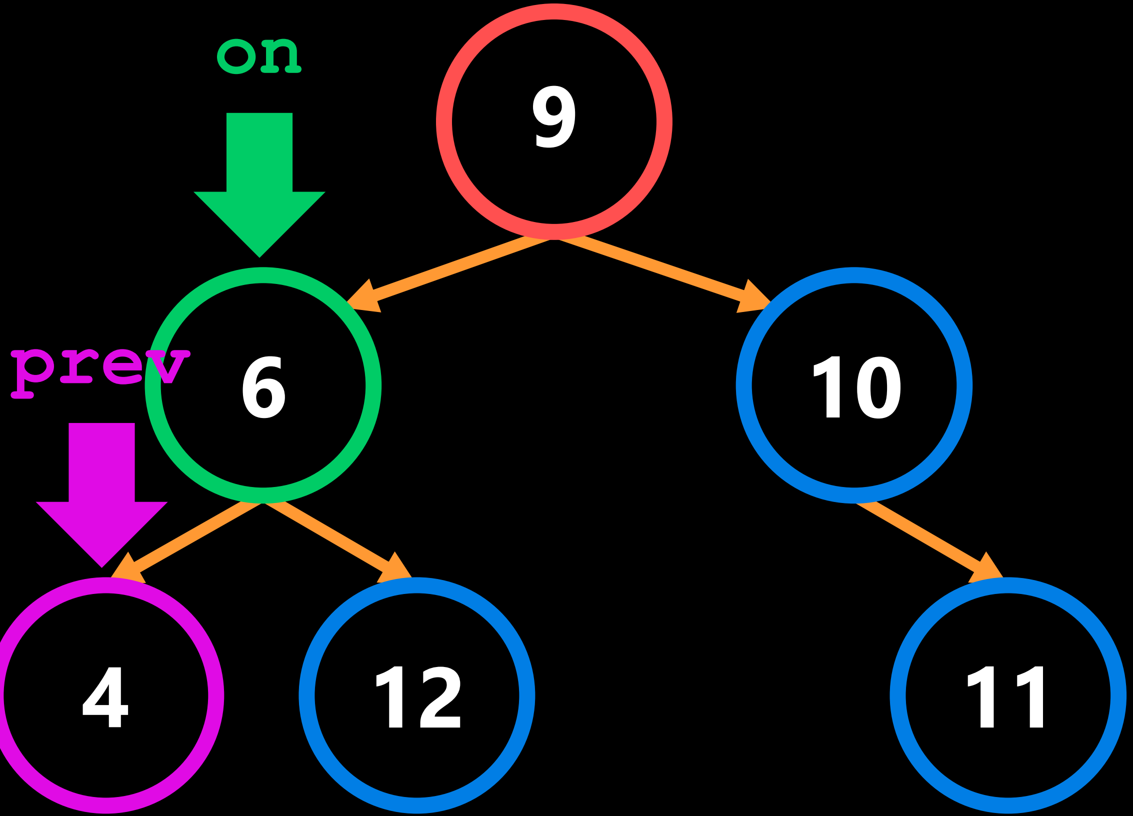
False

Set on to left node in stack.



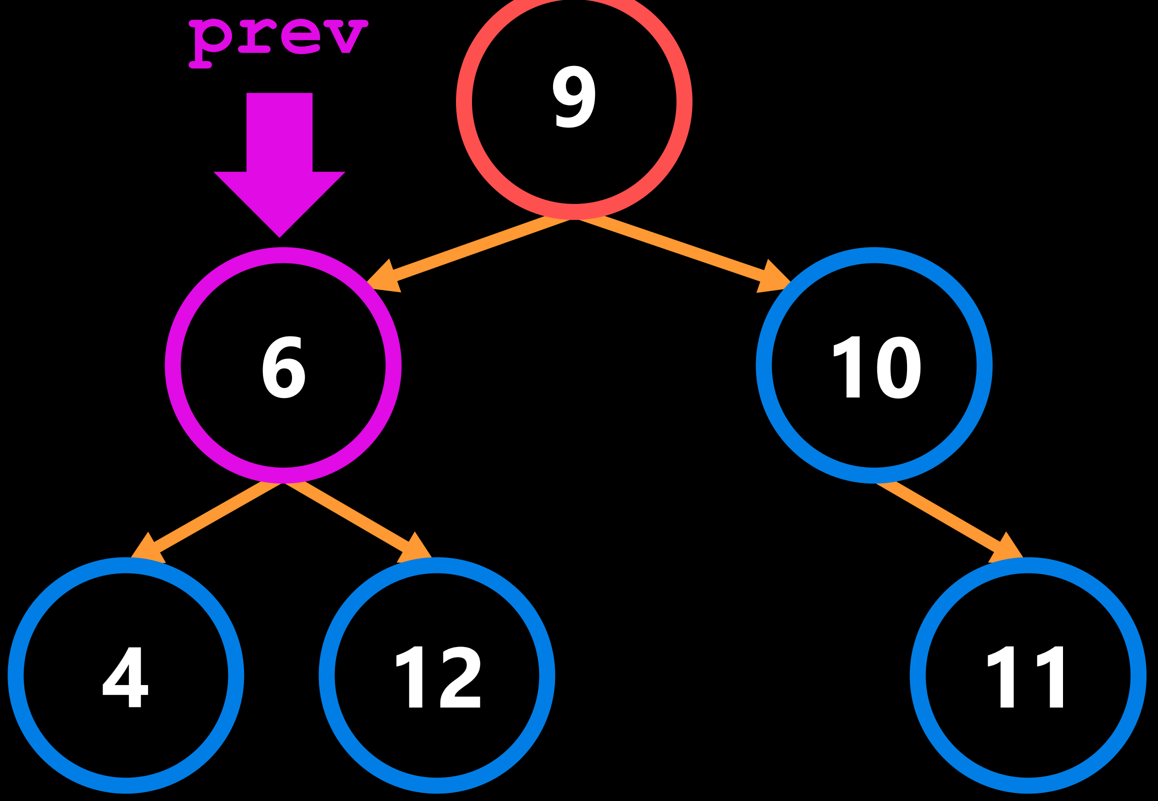
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:    True

            while on is not None:    False
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:    False
                return False

            prev = on
            on = on.right

        return True
```

**stack = [9]**



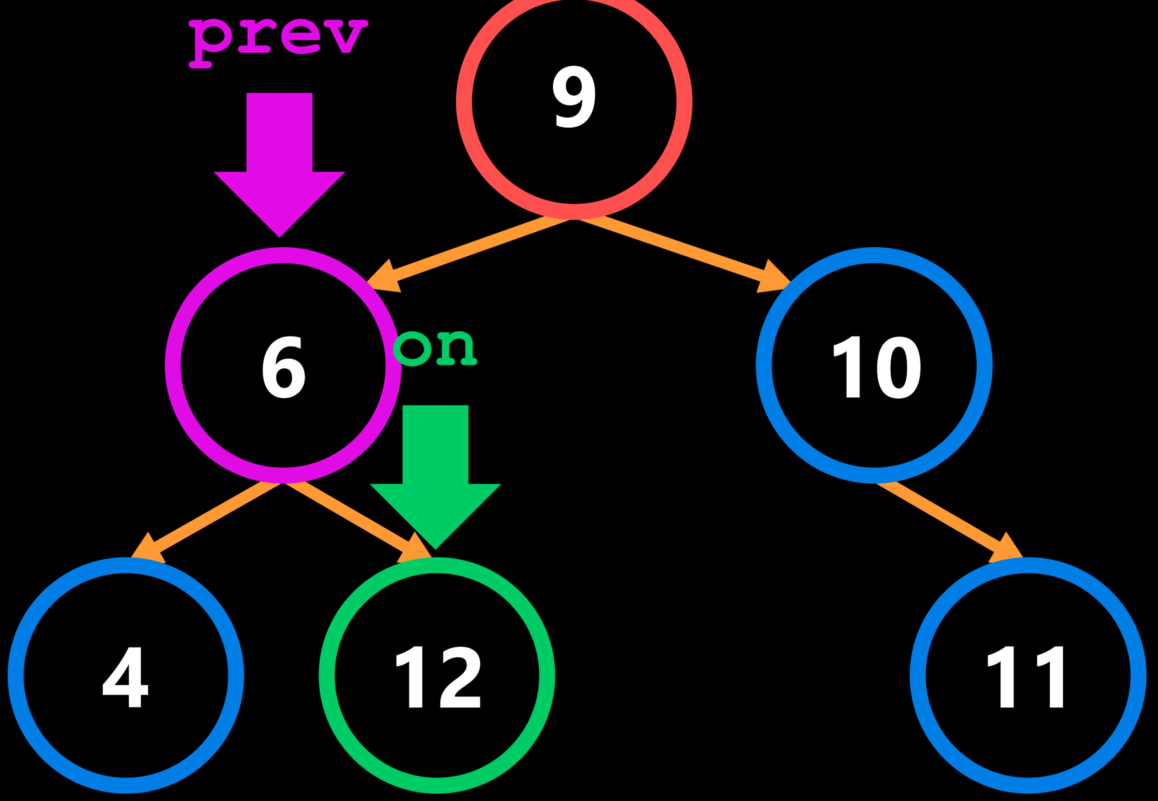
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:        True

            while on is not None:        False
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:        False
                return False

            prev = on        Set prev to on.
            on = on.right

        return True
```

stack = [9]



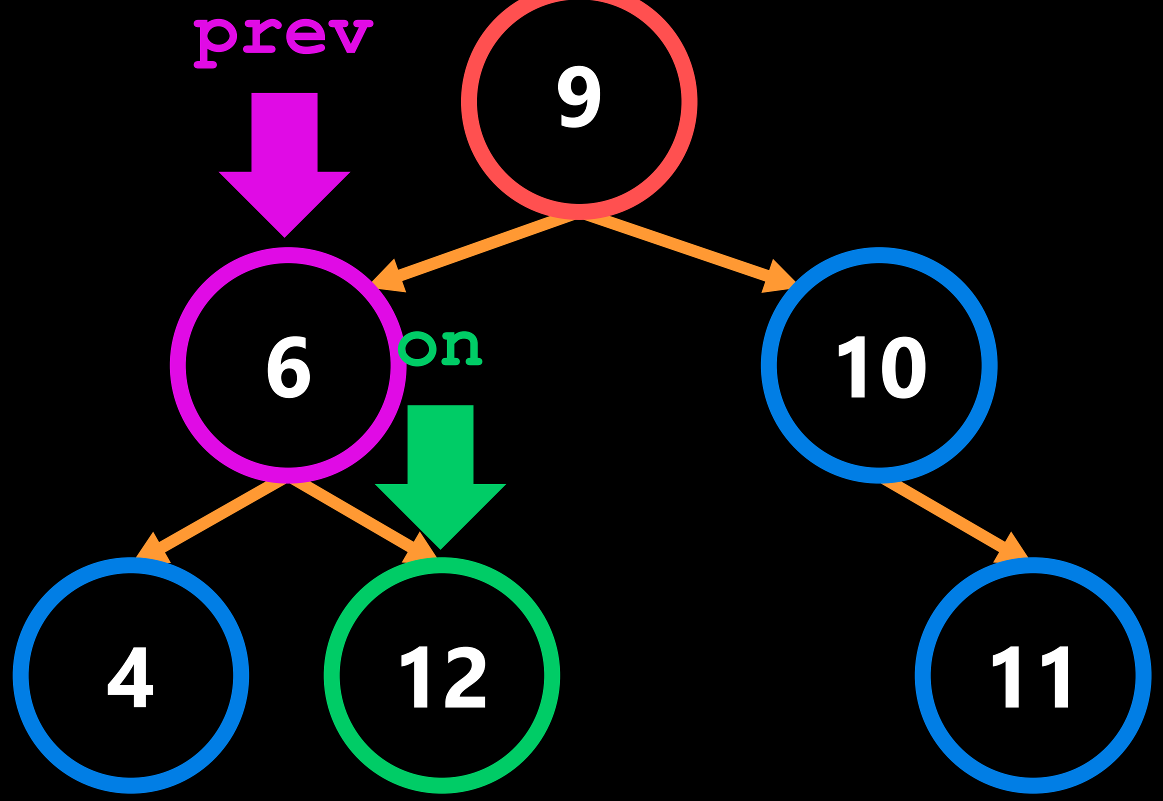
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:  ← True

            while on is not None:  ← False
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:  ← False
                return False

            prev = on
            on = on.right  ← Move on to the right pointer.

        return True
```

stack = [9]



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:  ← True

            while on is not None:  ← True
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

stack = [9]

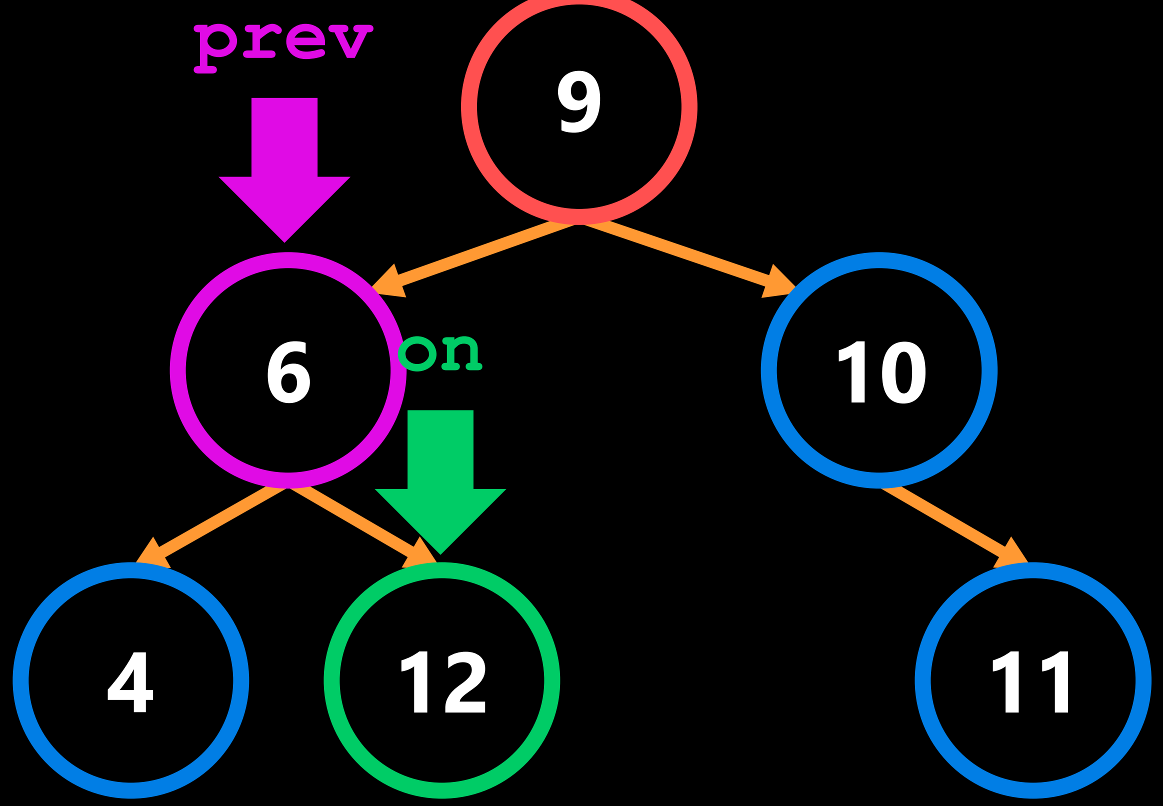


```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

True

True

stack = [ 9 ]



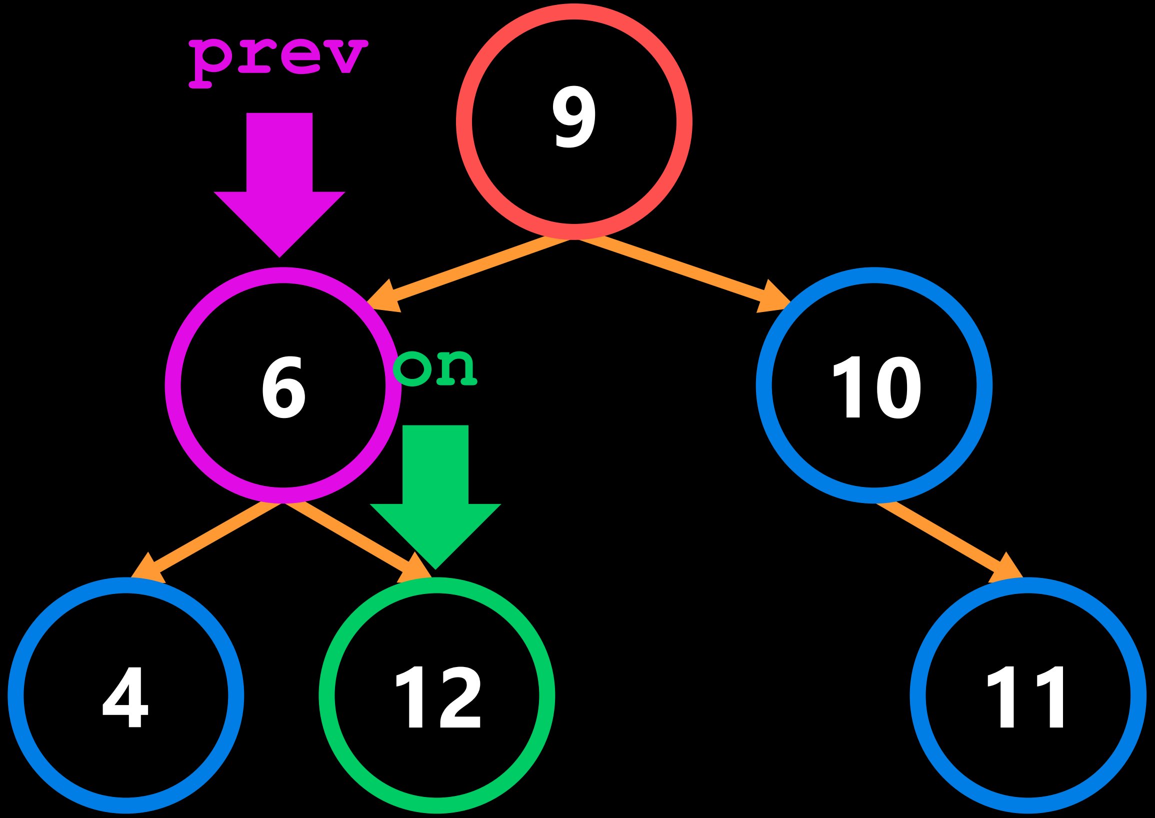
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

True

True

Add on to stack.

stack = [ 9 12 ]



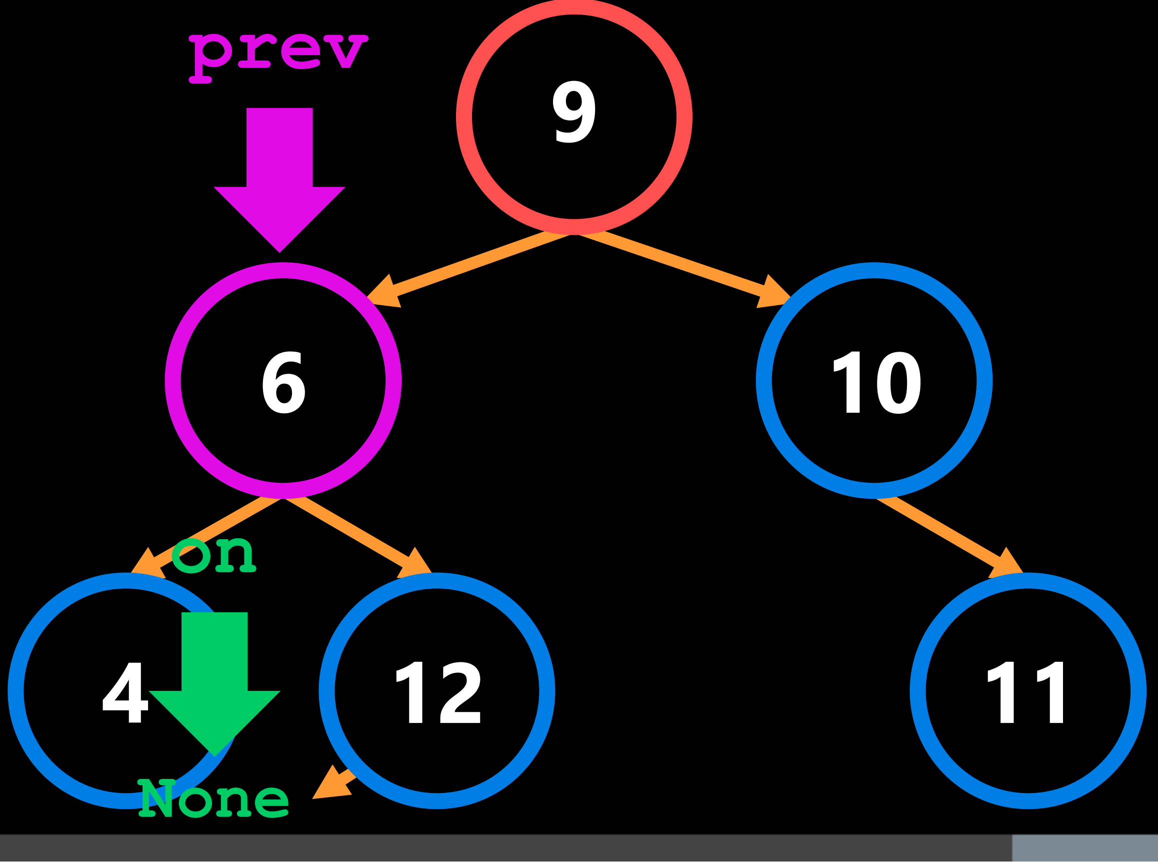
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

**stack = [ 9 12 ]**

True

True

**Move on to left node pointer.**



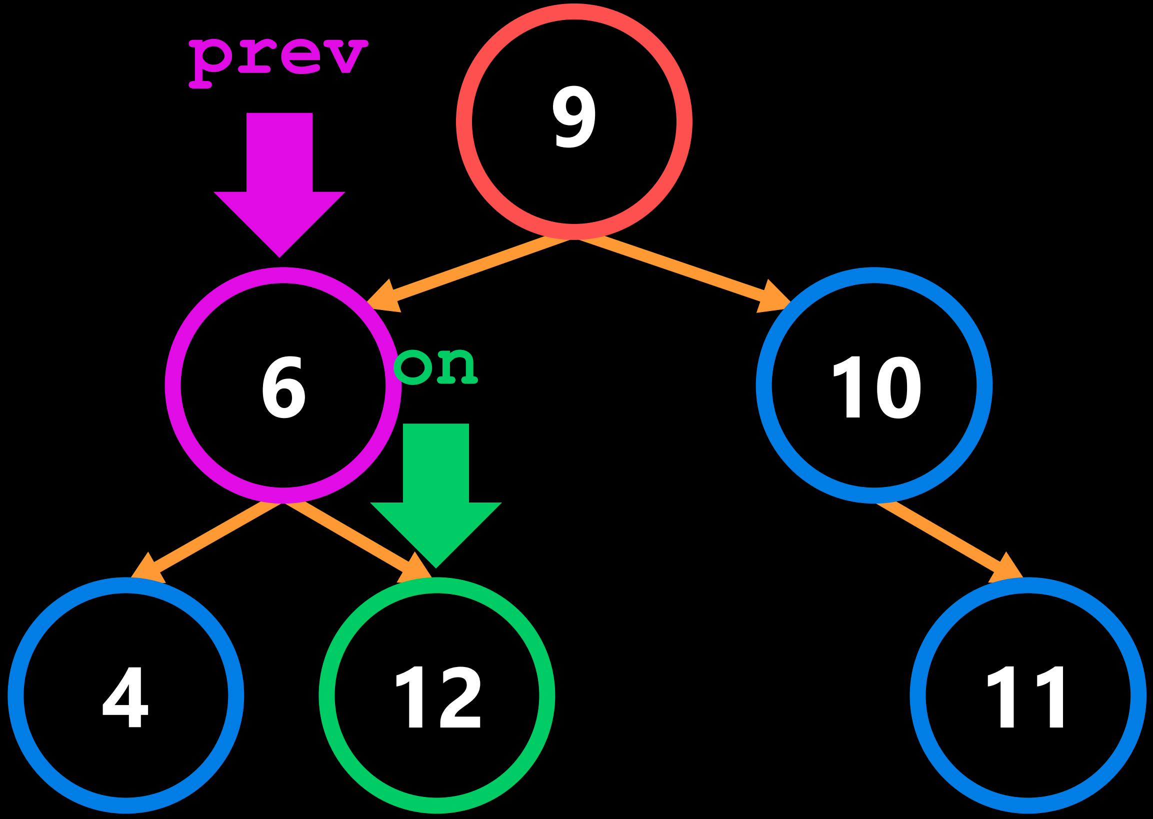
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

**stack = [ 9 ]**

True

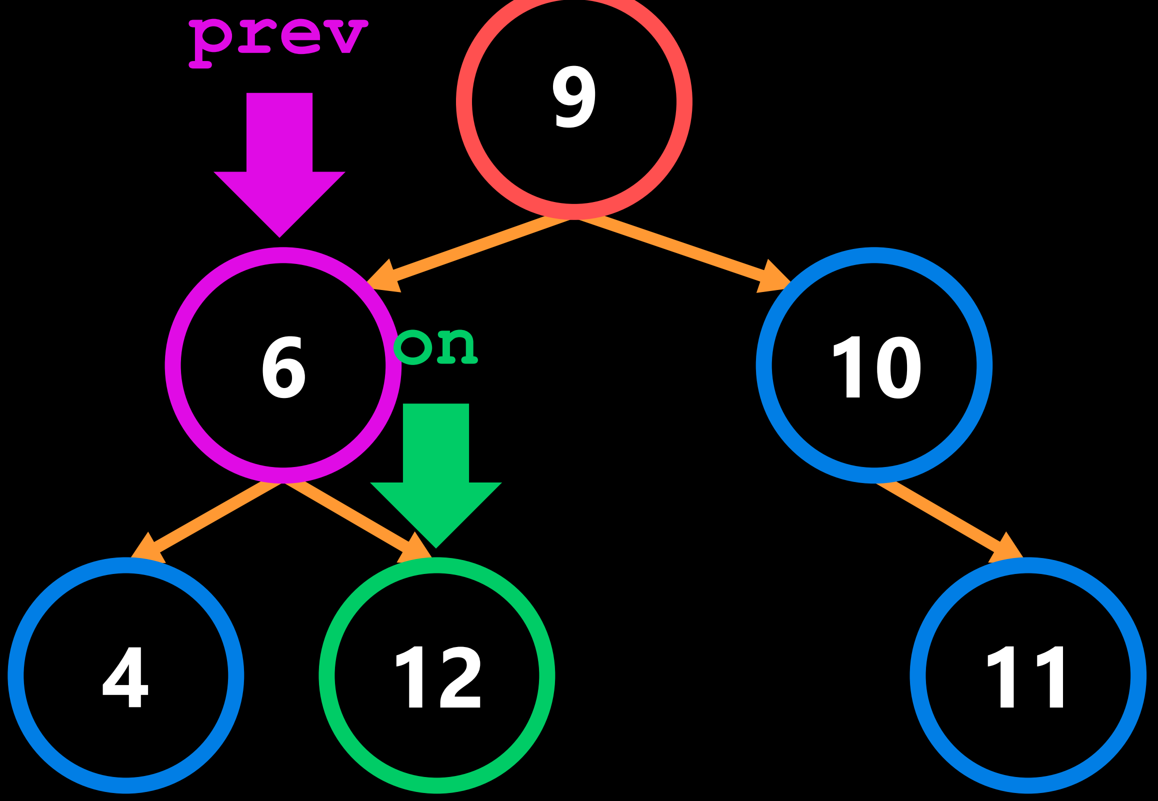
True

**Set on to left node in stack.**



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:  ← True

            while on is not None:  ← True
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:  ← False
                return False

            prev = on
            on = on.right

        return True
```

stack = [9]



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

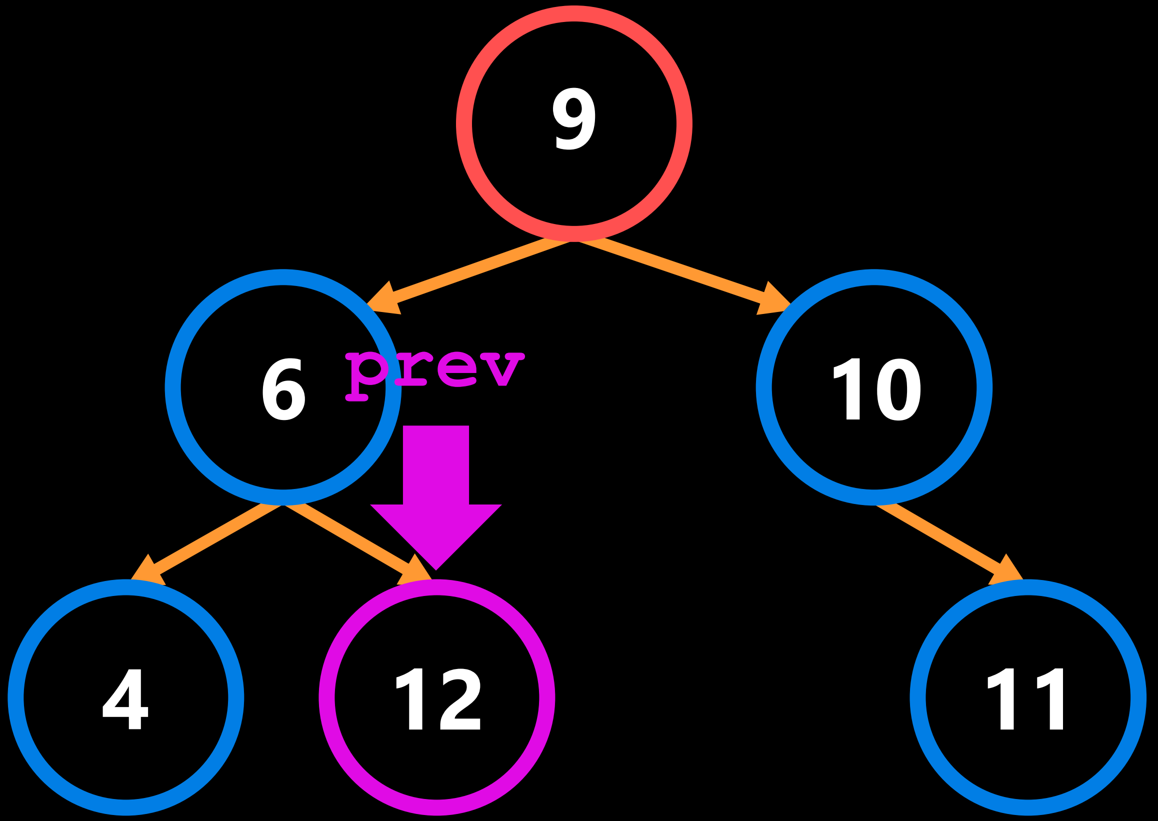
True

True

False

Set prev to on.

stack = [9]



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

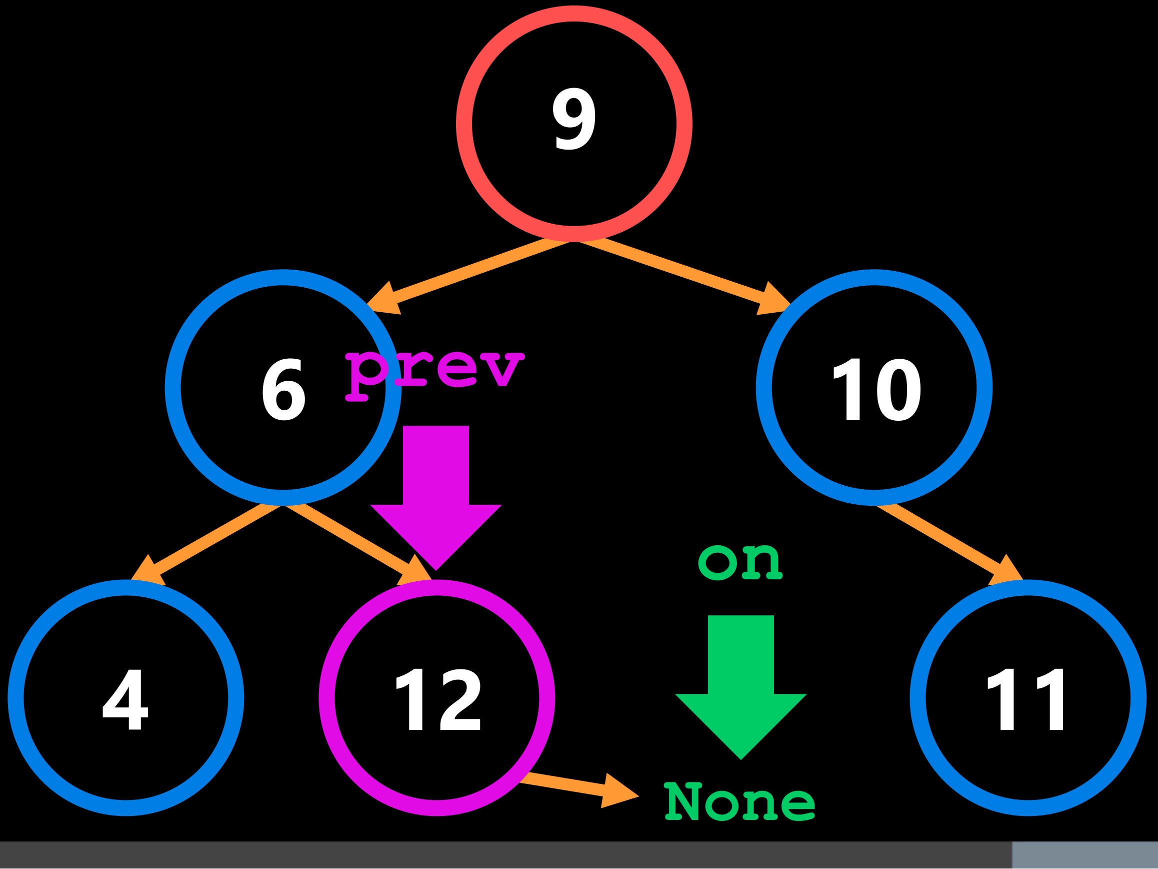
True

True

False

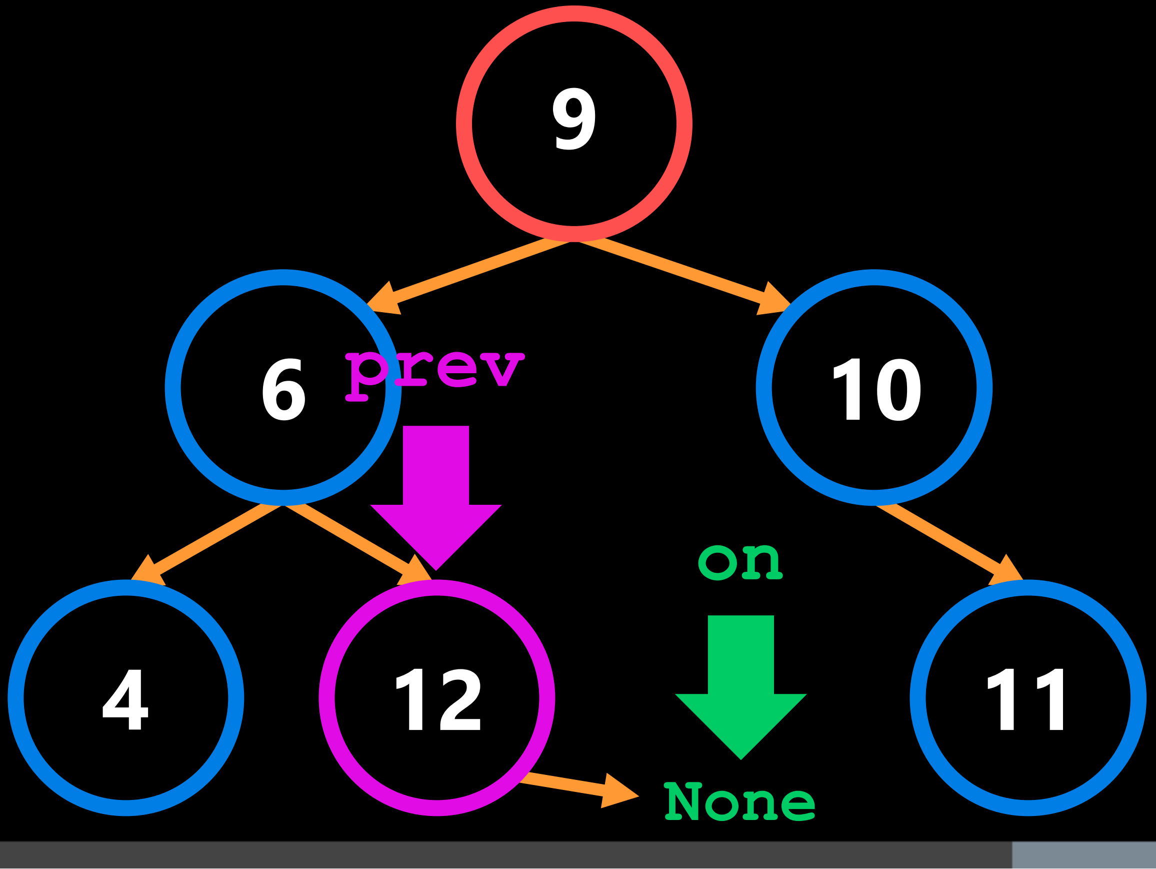
**Move on to the right pointer.**

**stack = [9]**



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:  ⬅ True

            while on is not None:  ⬅ False
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

stack = [9]



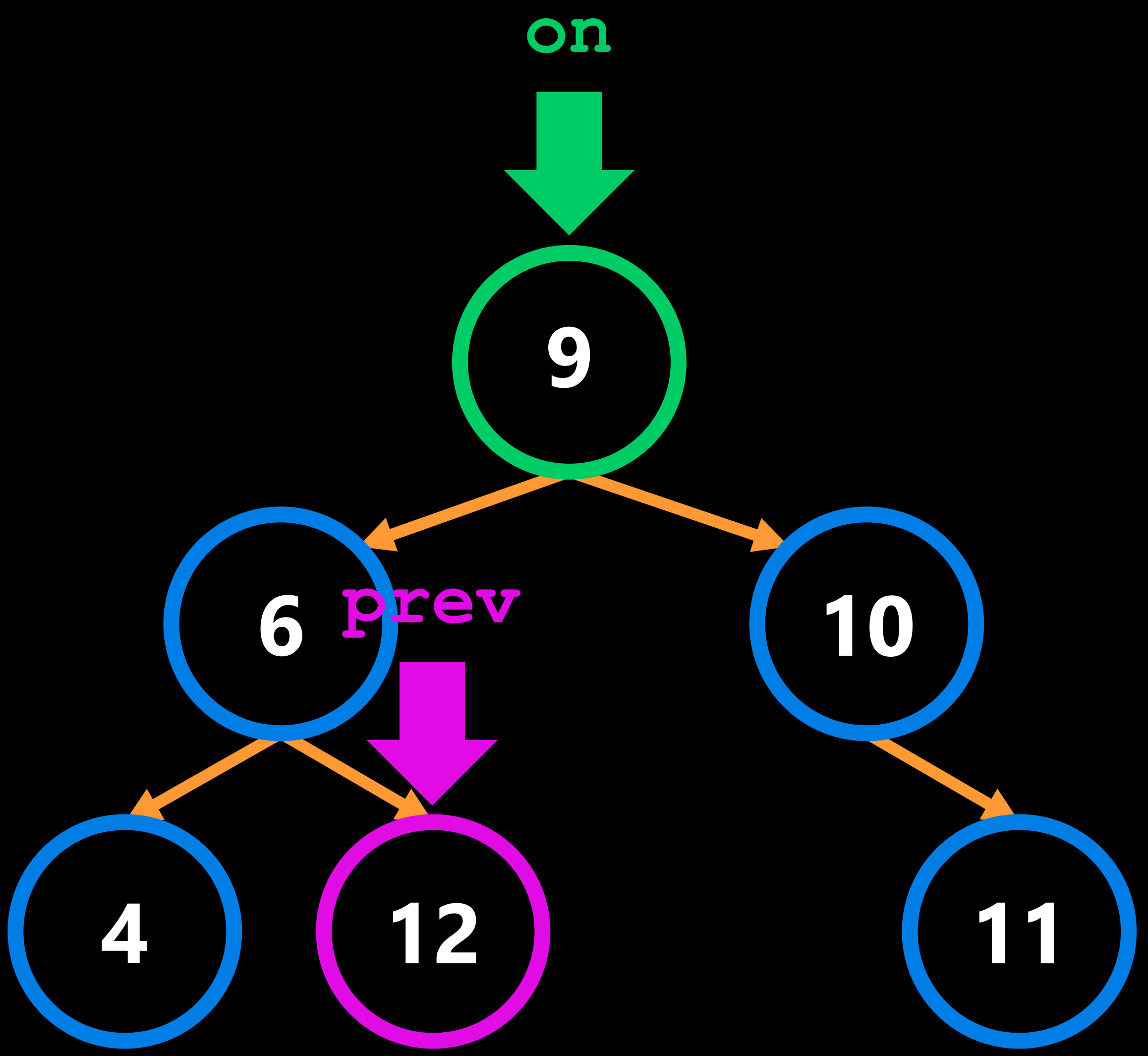
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

True

False

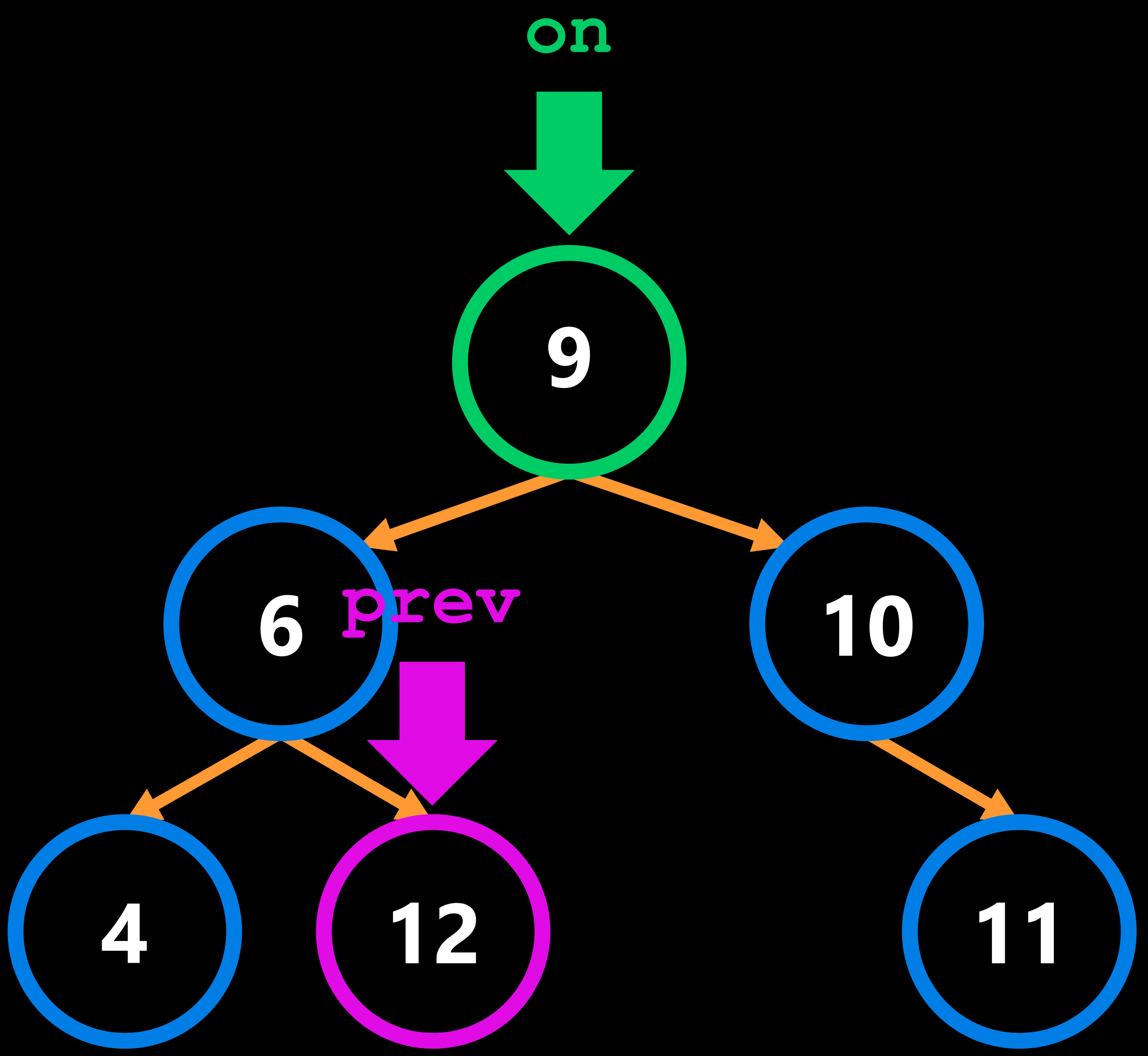
stack = []

Set on to left node in stack.



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:  ← True

            while on is not None:  ← False
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:  ← True
                return False

            prev = on
            on = on.right

        return True
```

stack = [ ]



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""

    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root

    def print_tree(self): ...

    def is_valid(self):
        """
        (self) -> NoneType
        Checks if self.root is a valid binary search tree.
        """
        on = self.root
        stack = []
        prev = None

        while len(stack) > 0 or on is not None:

            while on is not None:
                stack.append(on)
                on = on.left

            on = stack.pop()

            if prev is not None and on.cargo <= prev.cargo:
                return False

            prev = on
            on = on.right

        return True
```

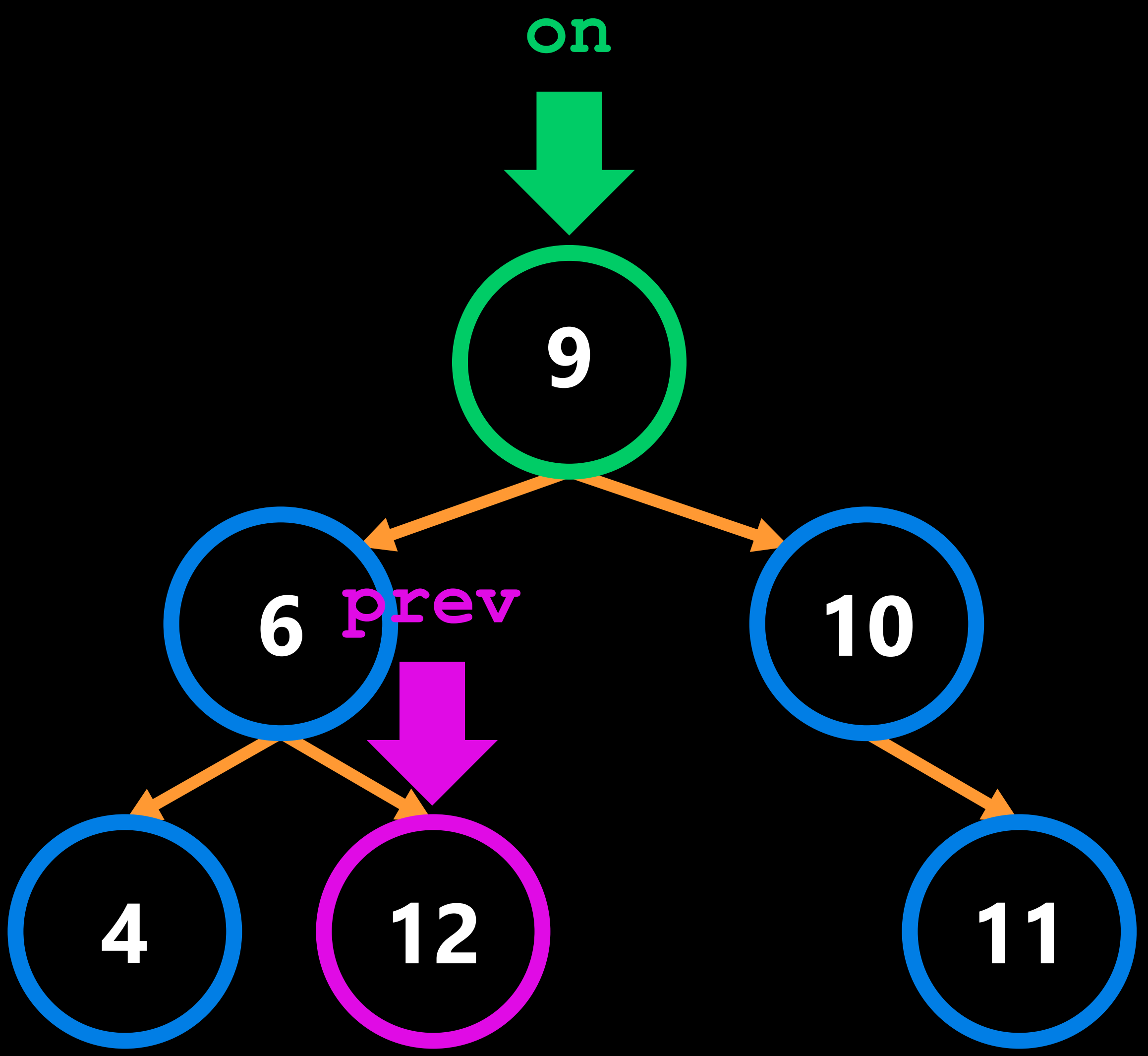
stack = [ ]

while len(stack) > 0 or on is not None: ← True

while on is not None: ← False

if prev is not None and on.cargo <= prev.cargo: ← True

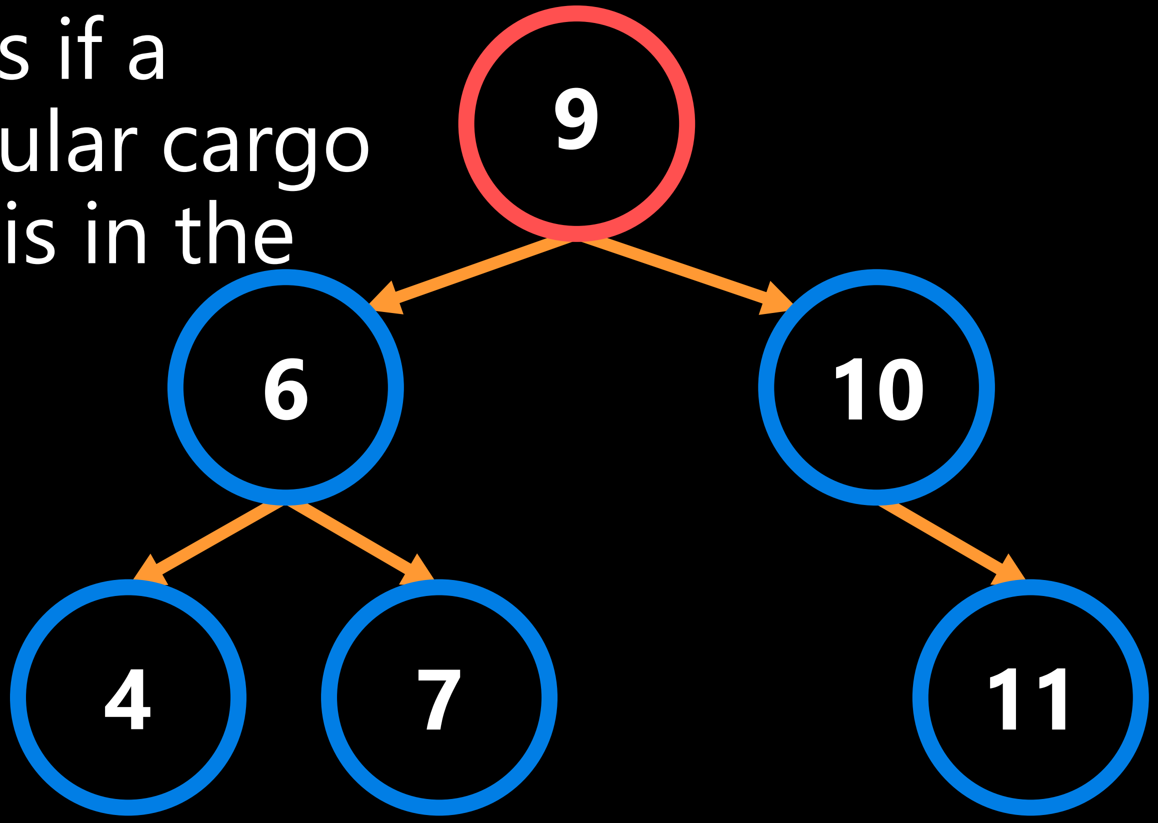
return False ← **Return False.**



# Breakout Session

- Create a new method that checks if a particular cargo value is in the tree.

```
tree.find(14)
```



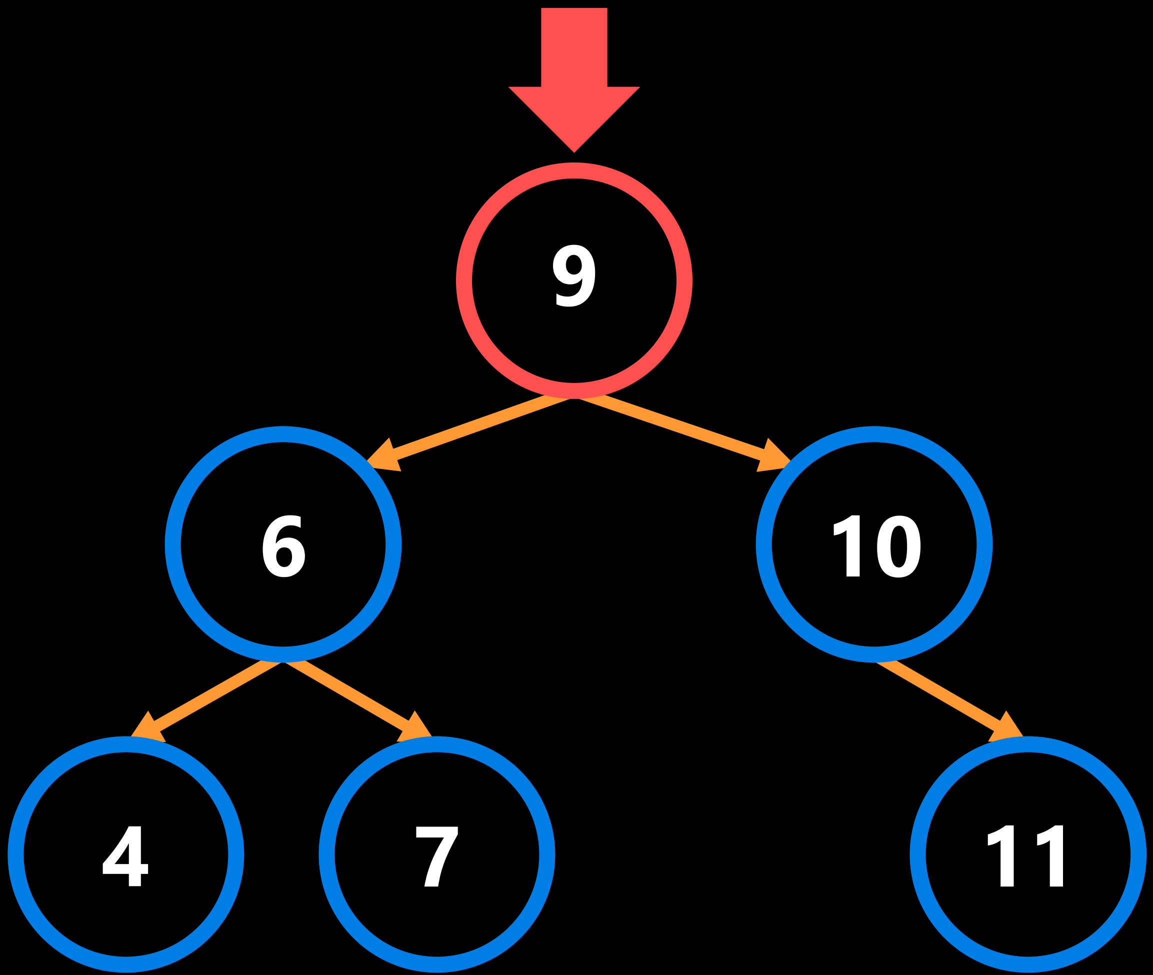
## Open your notebook

**Click Link:**
**3. Breakout Session**

```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:

            if cargo > on.cargo:
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```

tree.find(14)

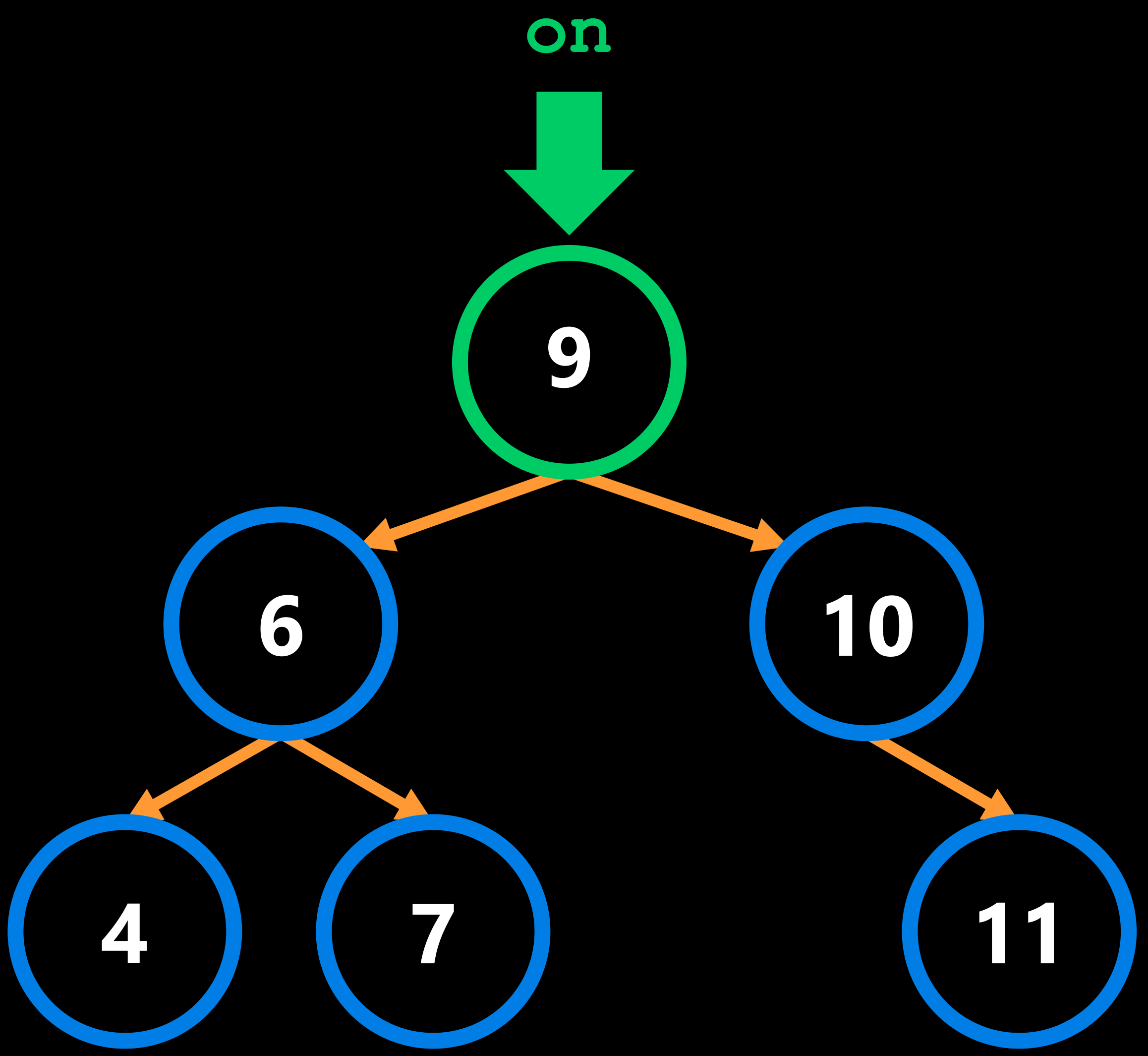
self.root



```python
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:

            if cargo > on.cargo:
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```

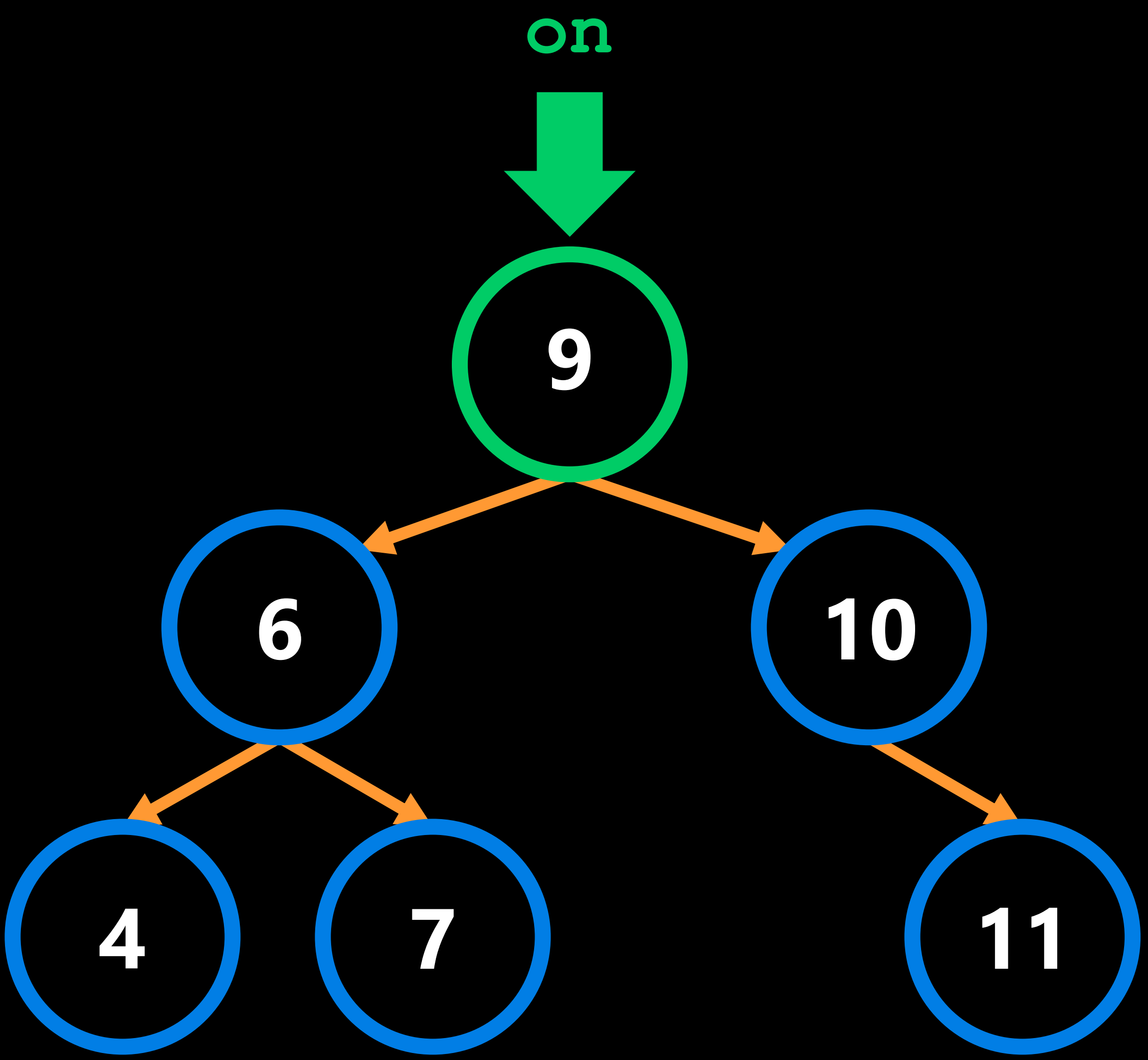
Set on position.

tree.find(14)



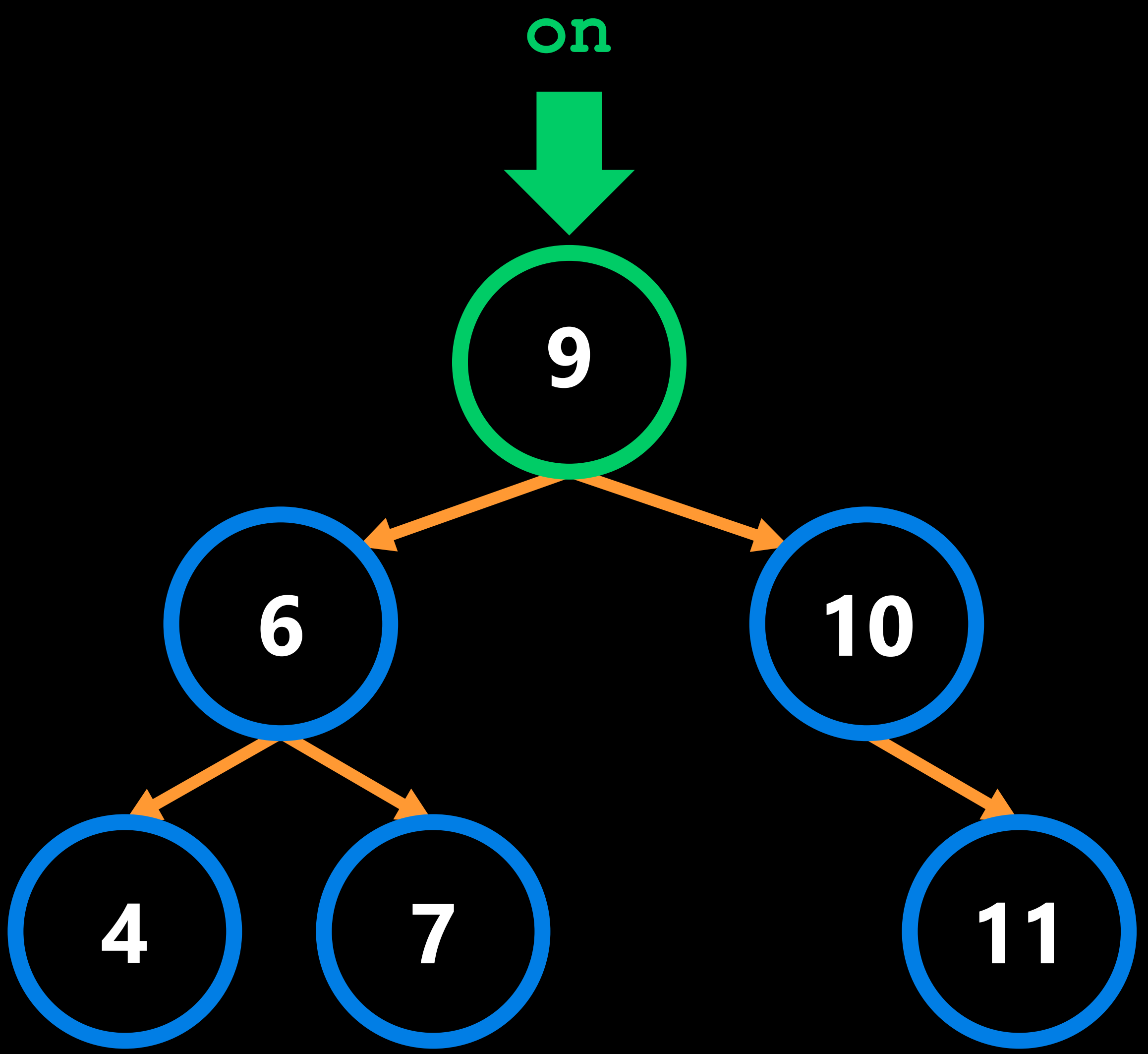
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:  ⬅ True

            if cargo > on.cargo:
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```

tree.find(14)



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:  ← True

            if cargo > on.cargo:  ← True  (14 > 9)
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```

tree.find(14)



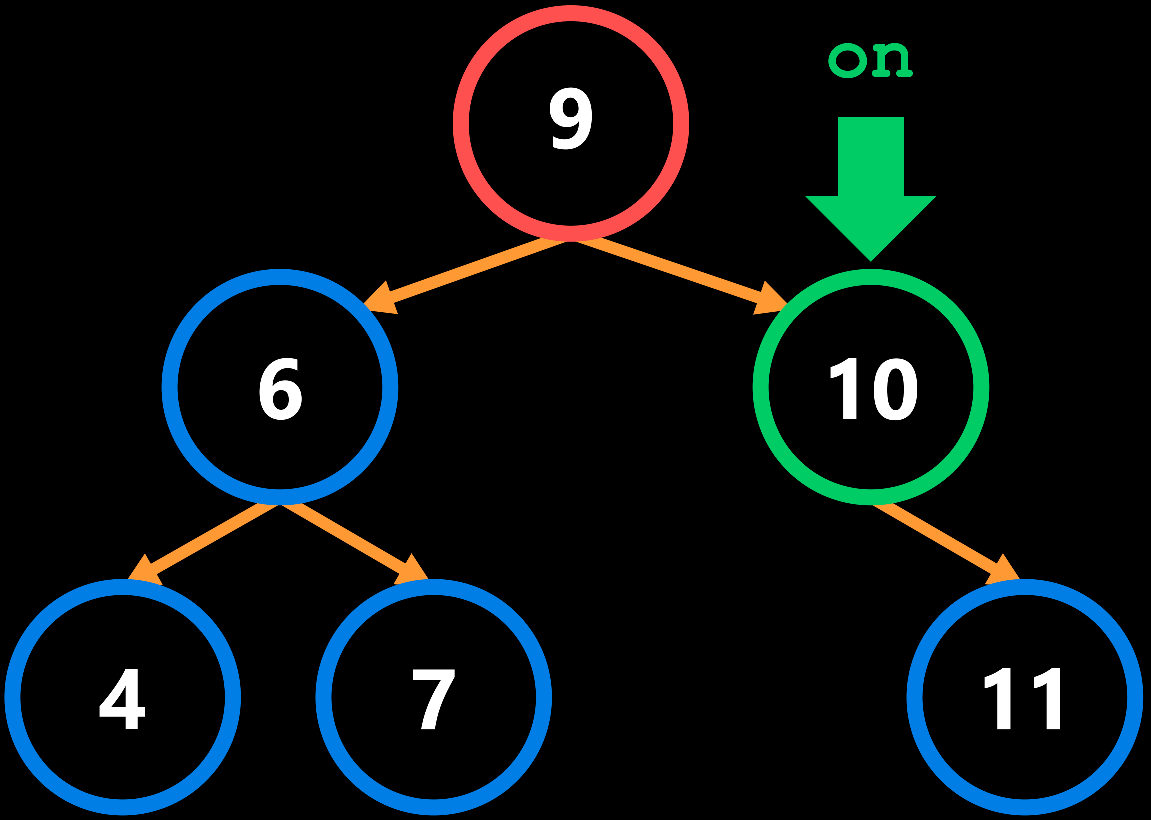
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:

            if cargo > on.cargo:
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```

True

True (14 > 9)

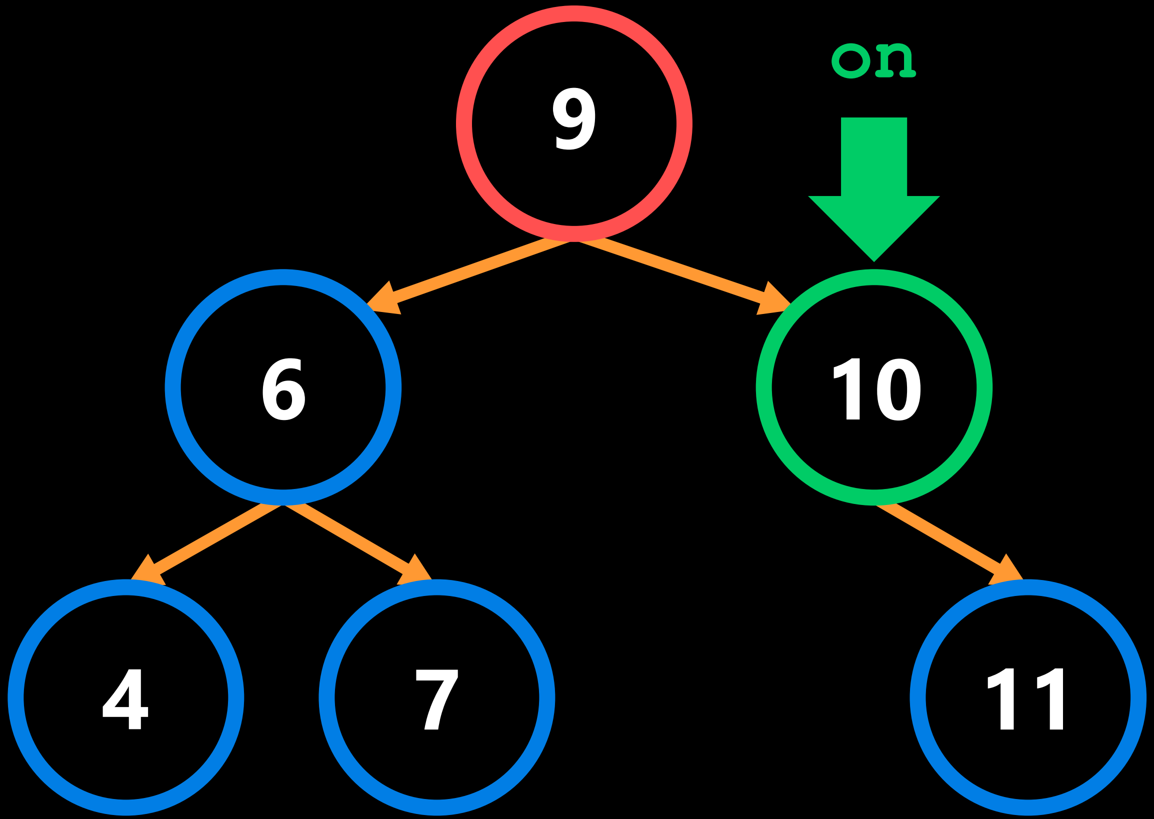
Move on to the right.

tree.find(14)



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:  ⬅ True

            if cargo > on.cargo:
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```

tree.find(14)

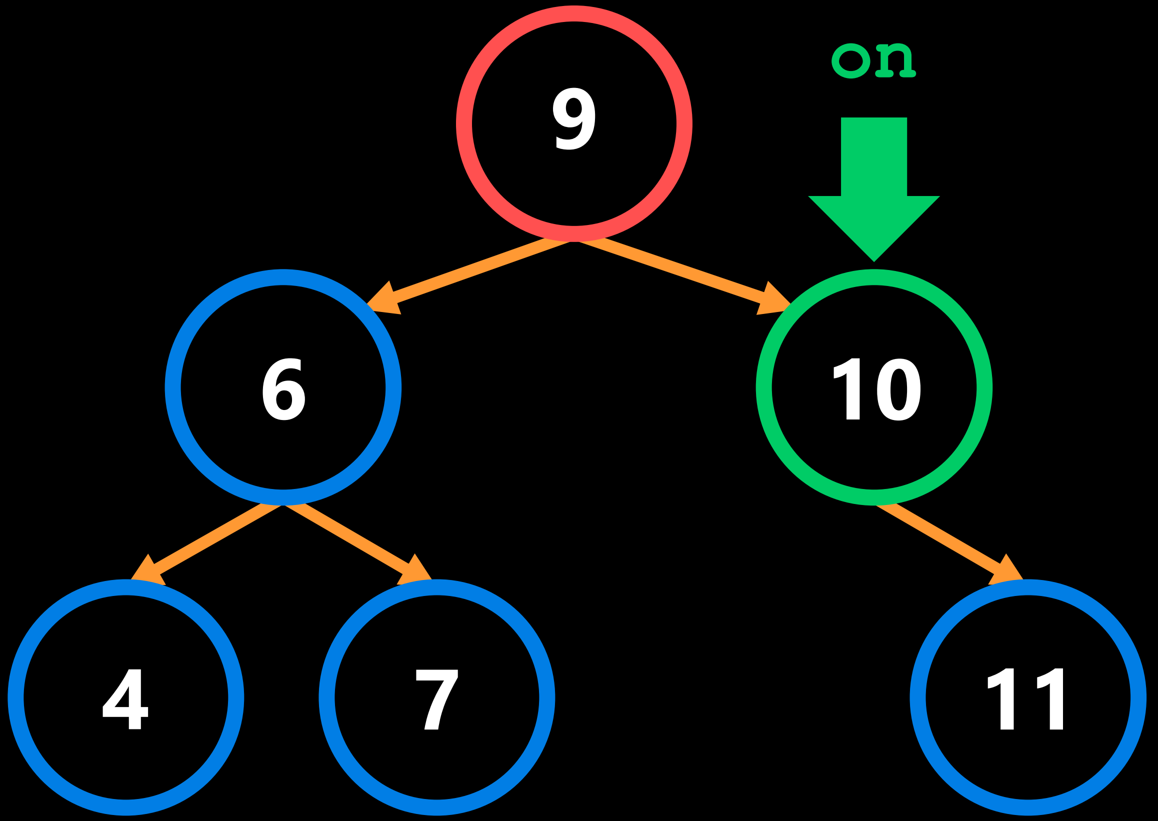


```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:

            if cargo > on.cargo:
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```

True

True (14 > 10)

tree.find(14)



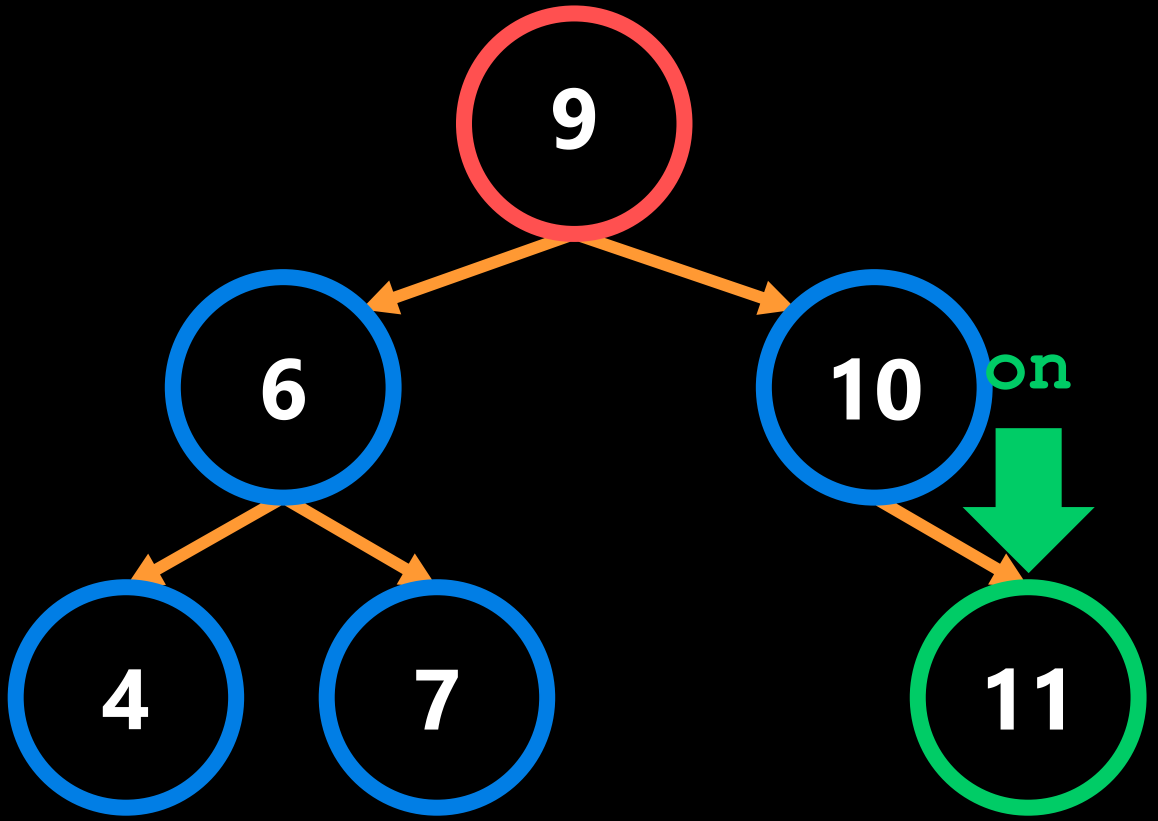
tree.find(14)

```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:

            if cargo > on.cargo:
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```

True

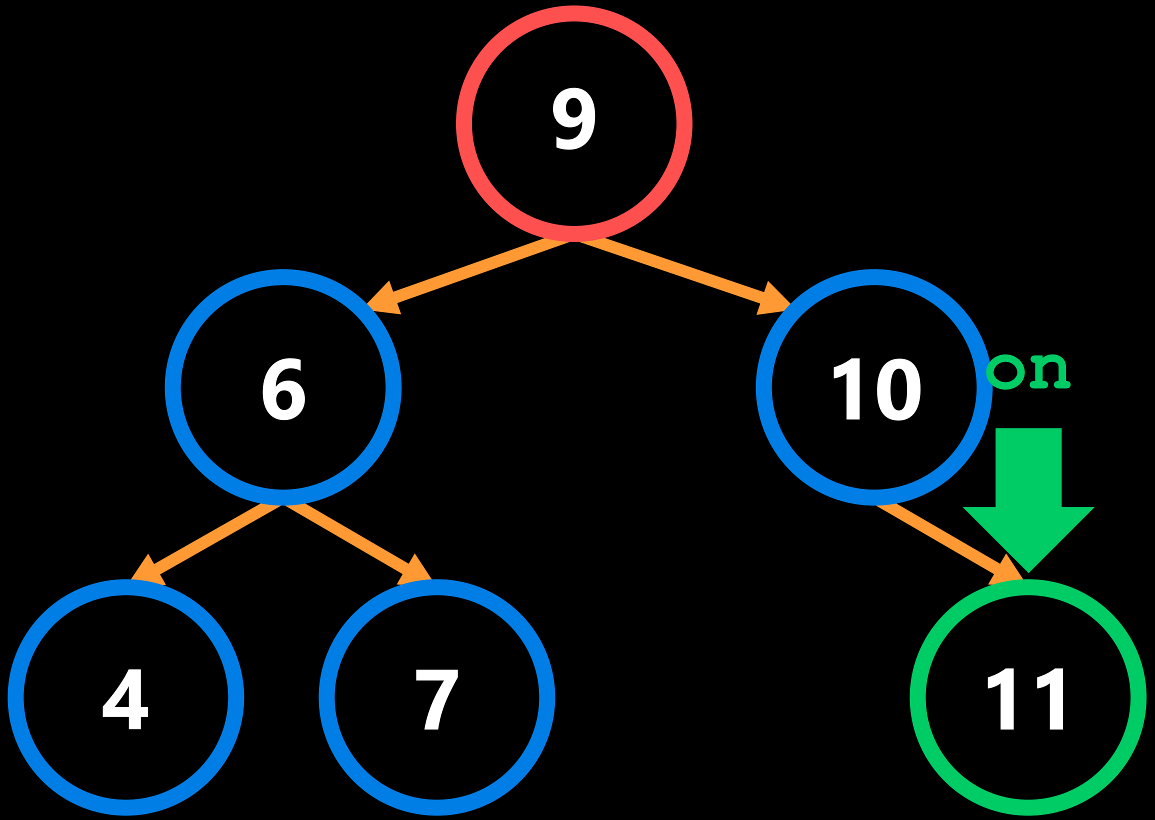
True (14 > 10)

Move on to the right.



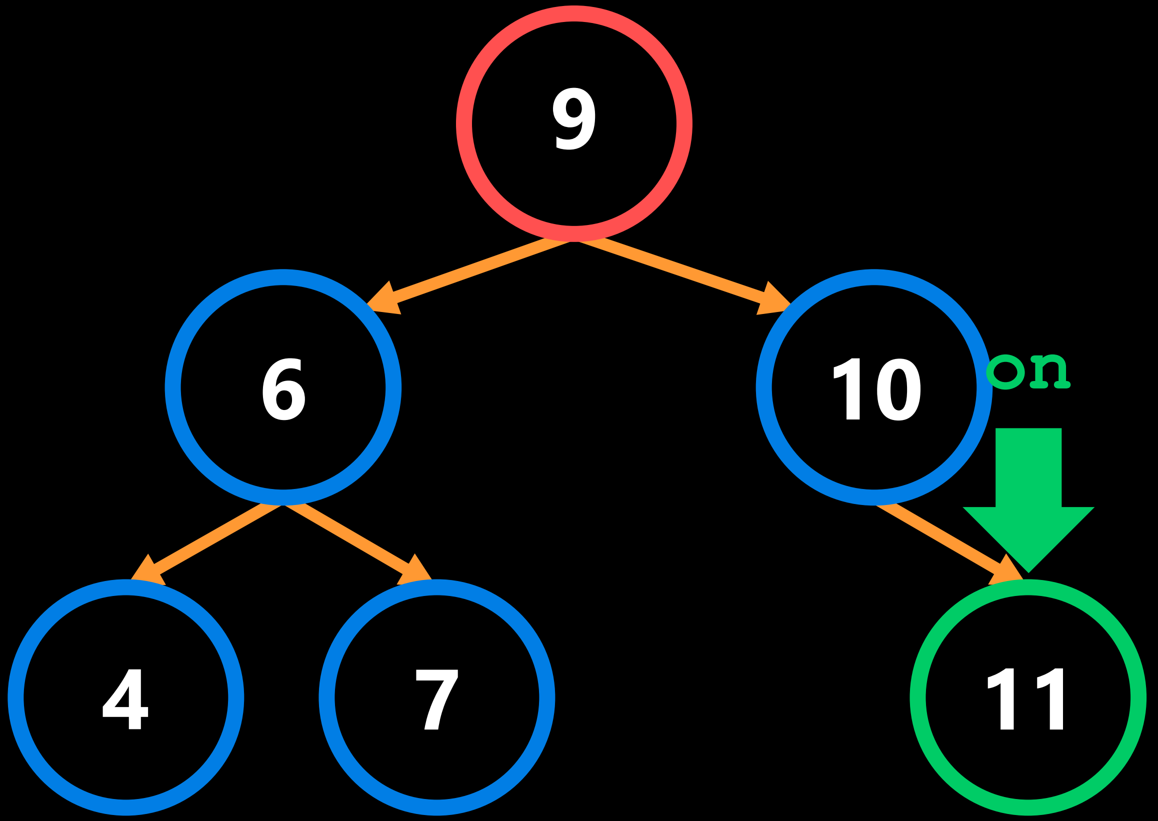
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:  ← True

            if cargo > on.cargo:
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```

tree.find(14)



tree.find(14)

```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:  ⬅ True

            if cargo > on.cargo:  ⬅ True  (14 > 11)
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```



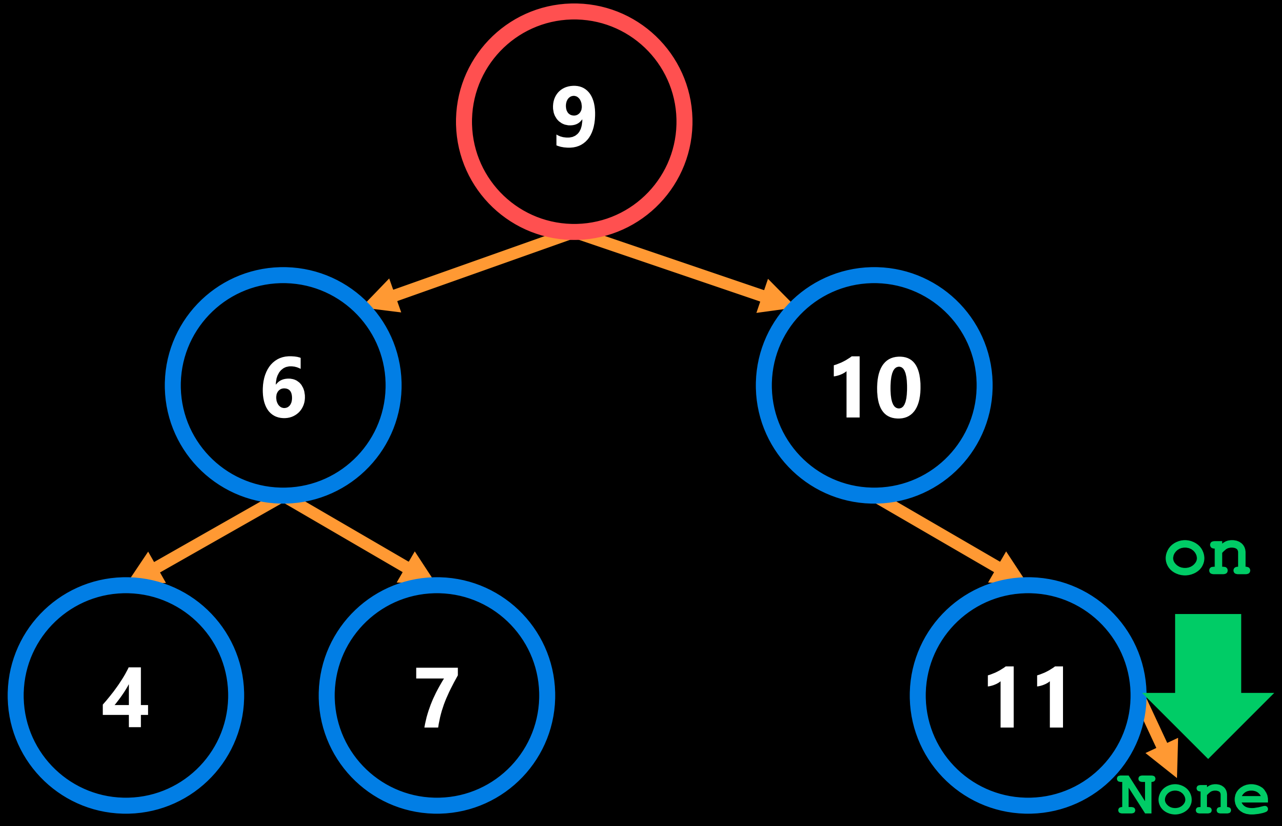
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:

            if cargo > on.cargo:
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```

True

True (14 > 11)

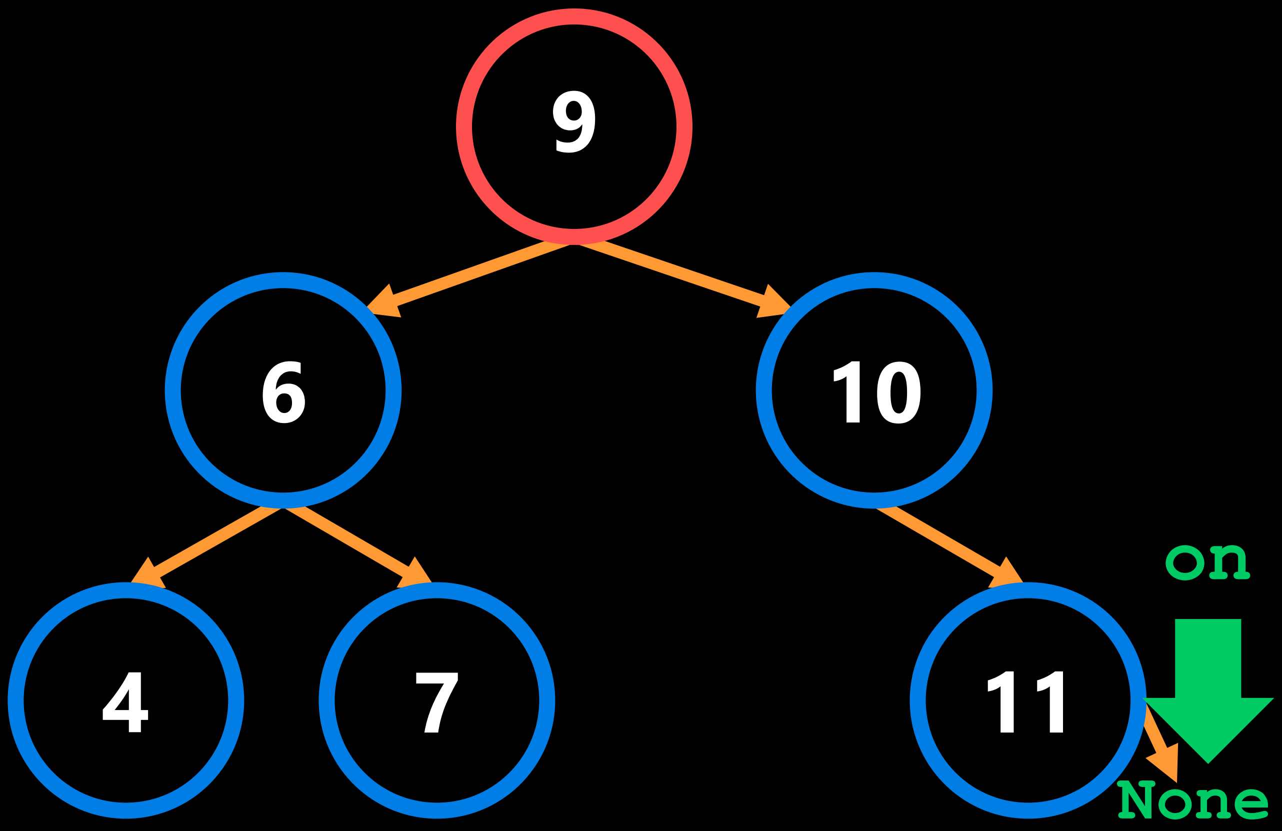
Move on to the right.

tree.find(14)



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:  ⇐ False

            if cargo > on.cargo:
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```

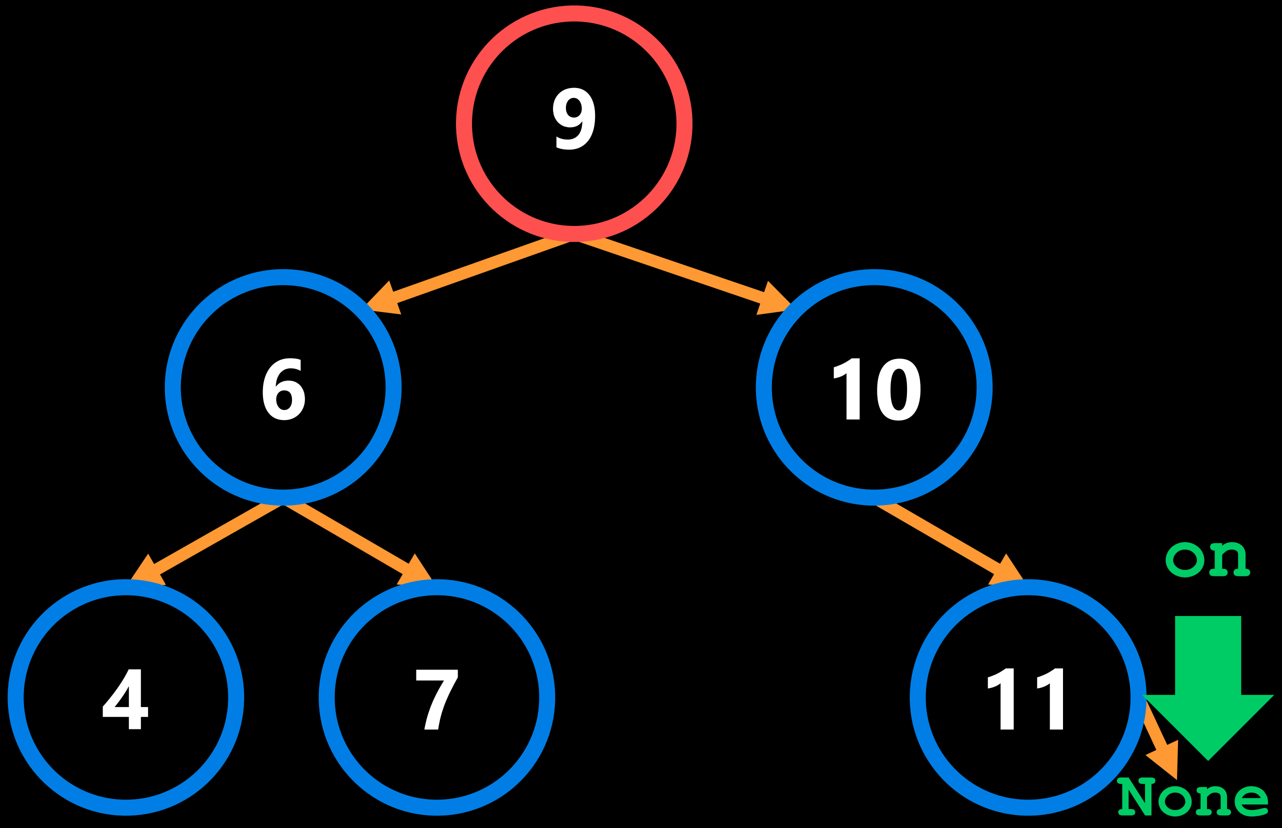
tree.find(14)



tree.find(14)

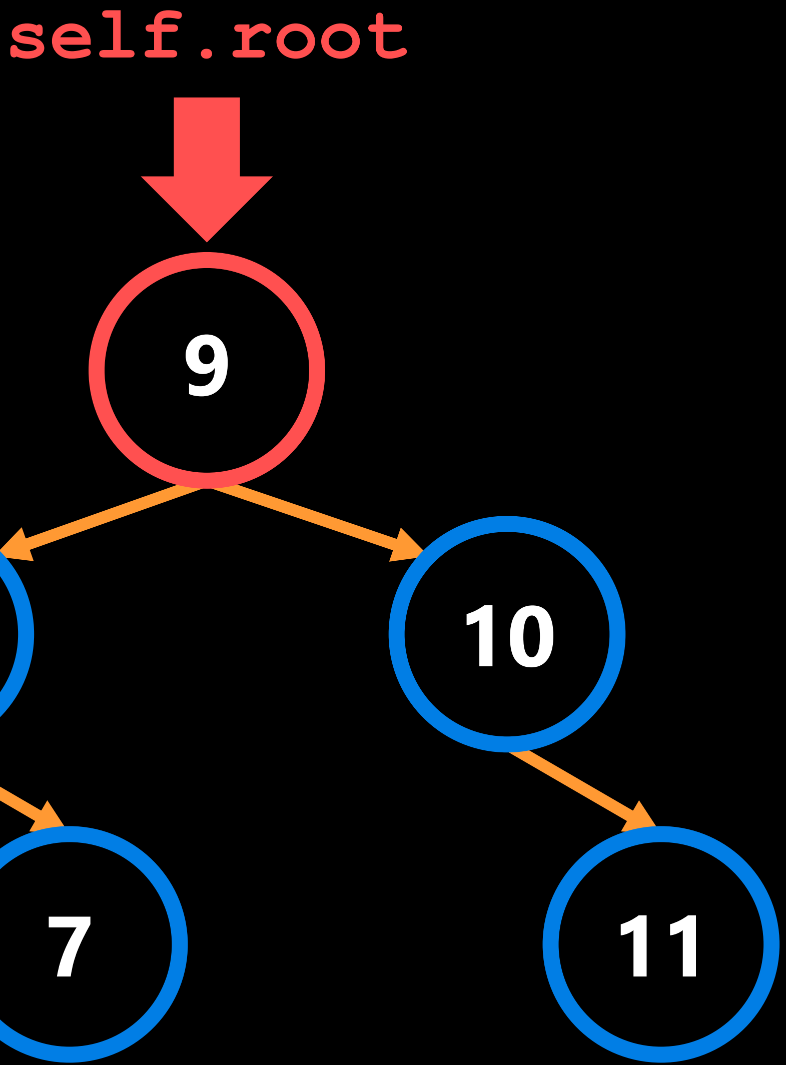
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:

            if cargo > on.cargo:
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```

**Didn't find the value.**



```python
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:

            if cargo > on.cargo:
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```

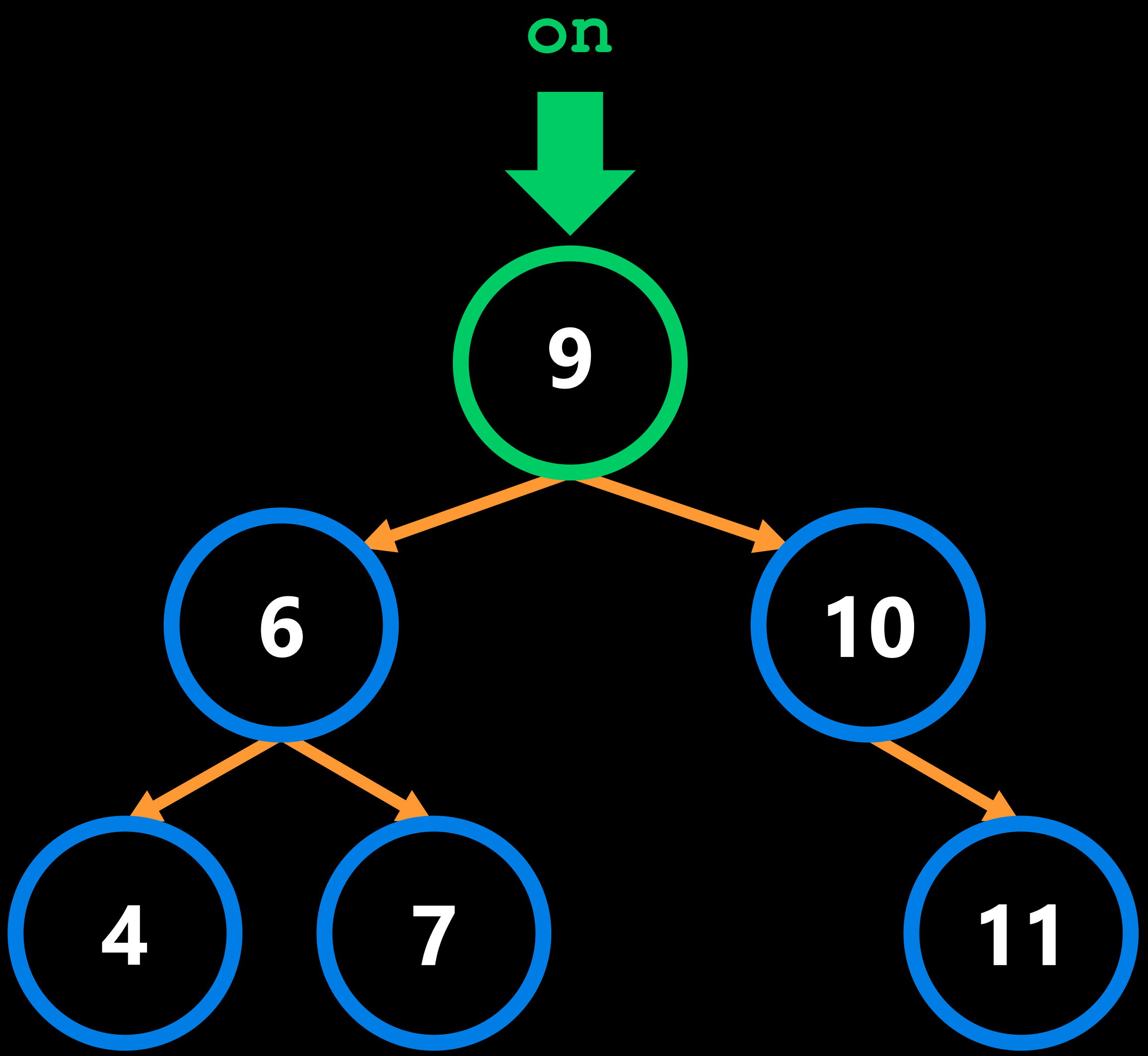
tree.find(4)



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:

            if cargo > on.cargo:
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```

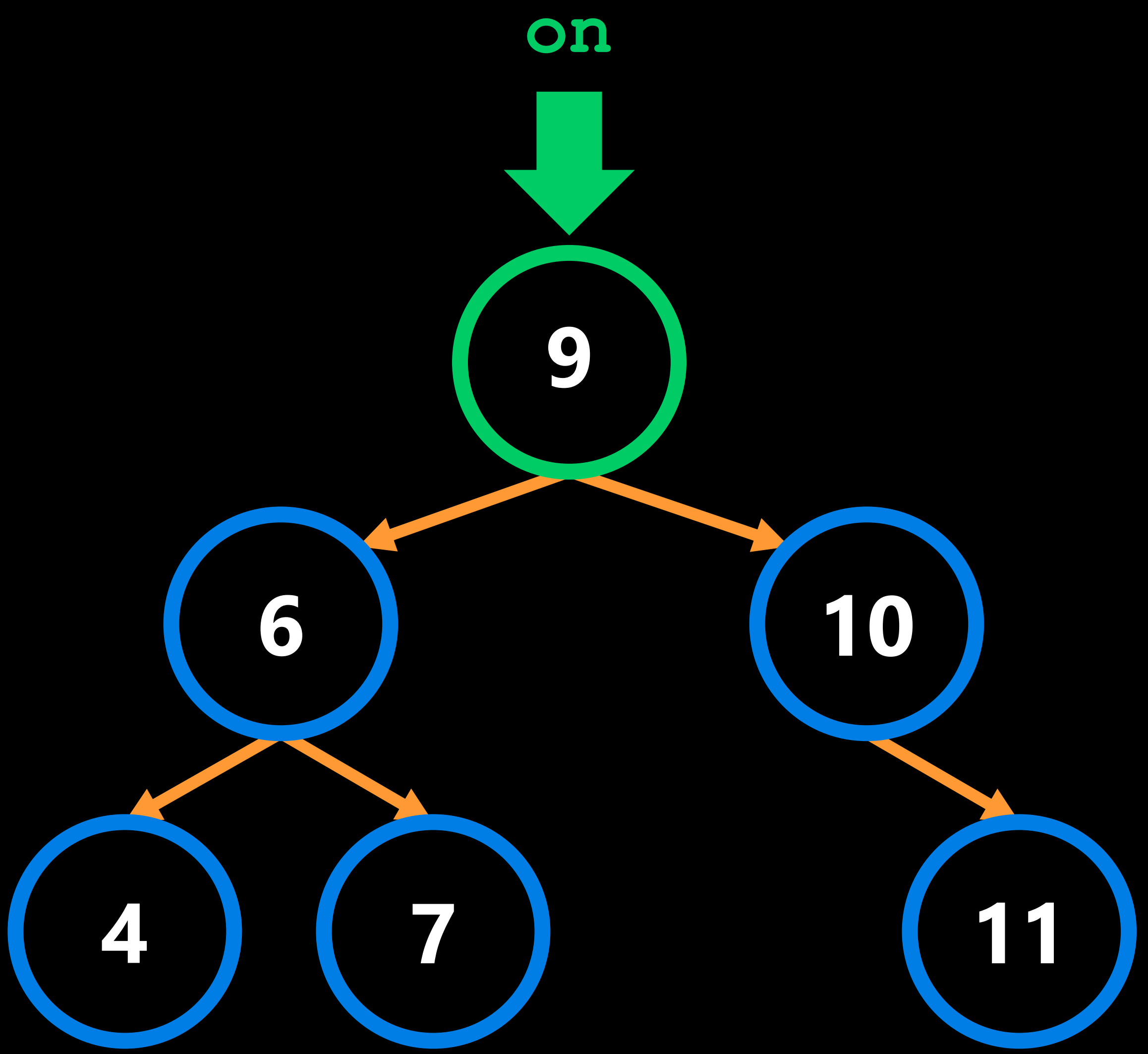
**Set on position.**

tree.find(4)



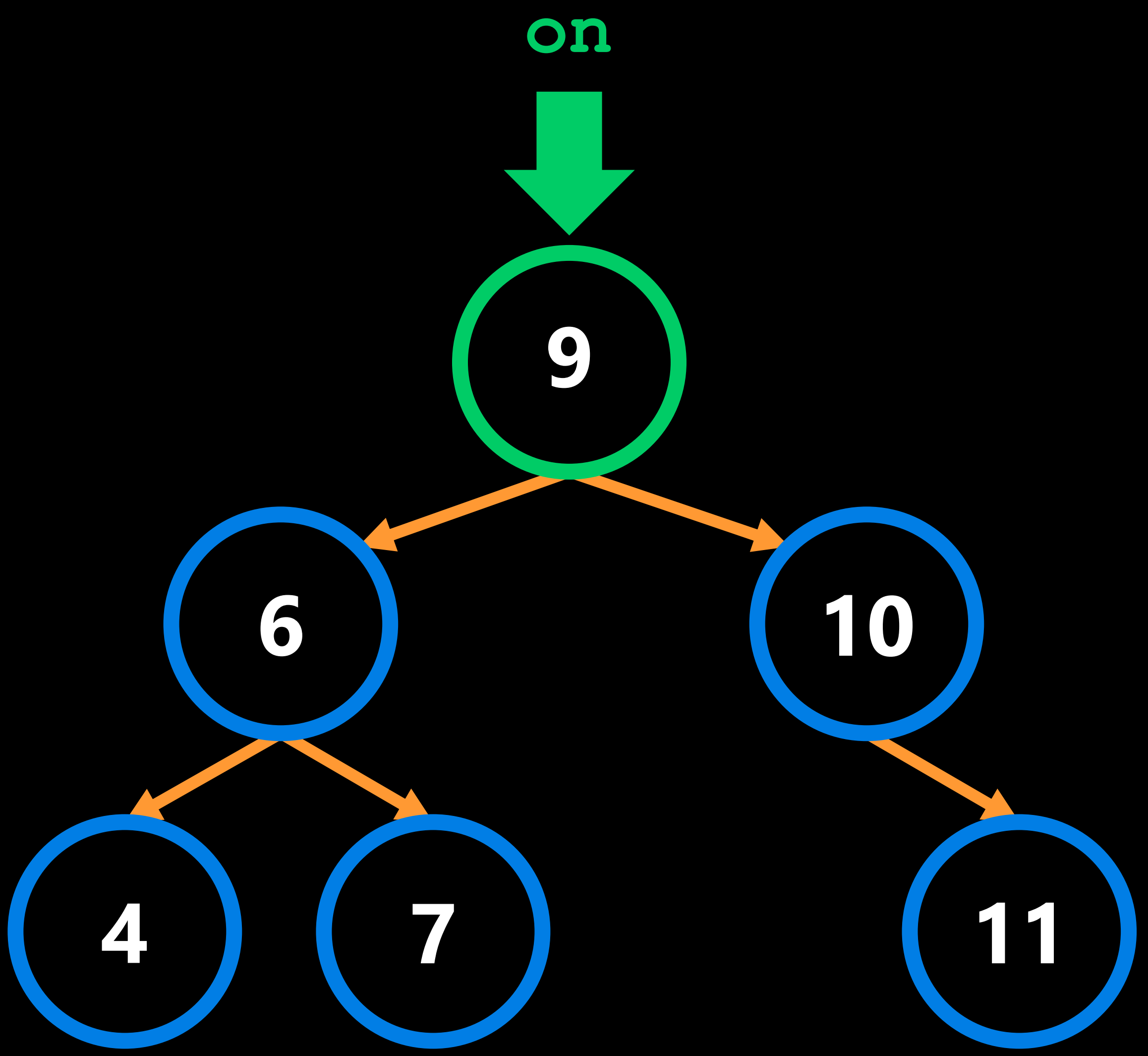
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:  ⬅ True

            if cargo > on.cargo:
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```

tree.find(4)



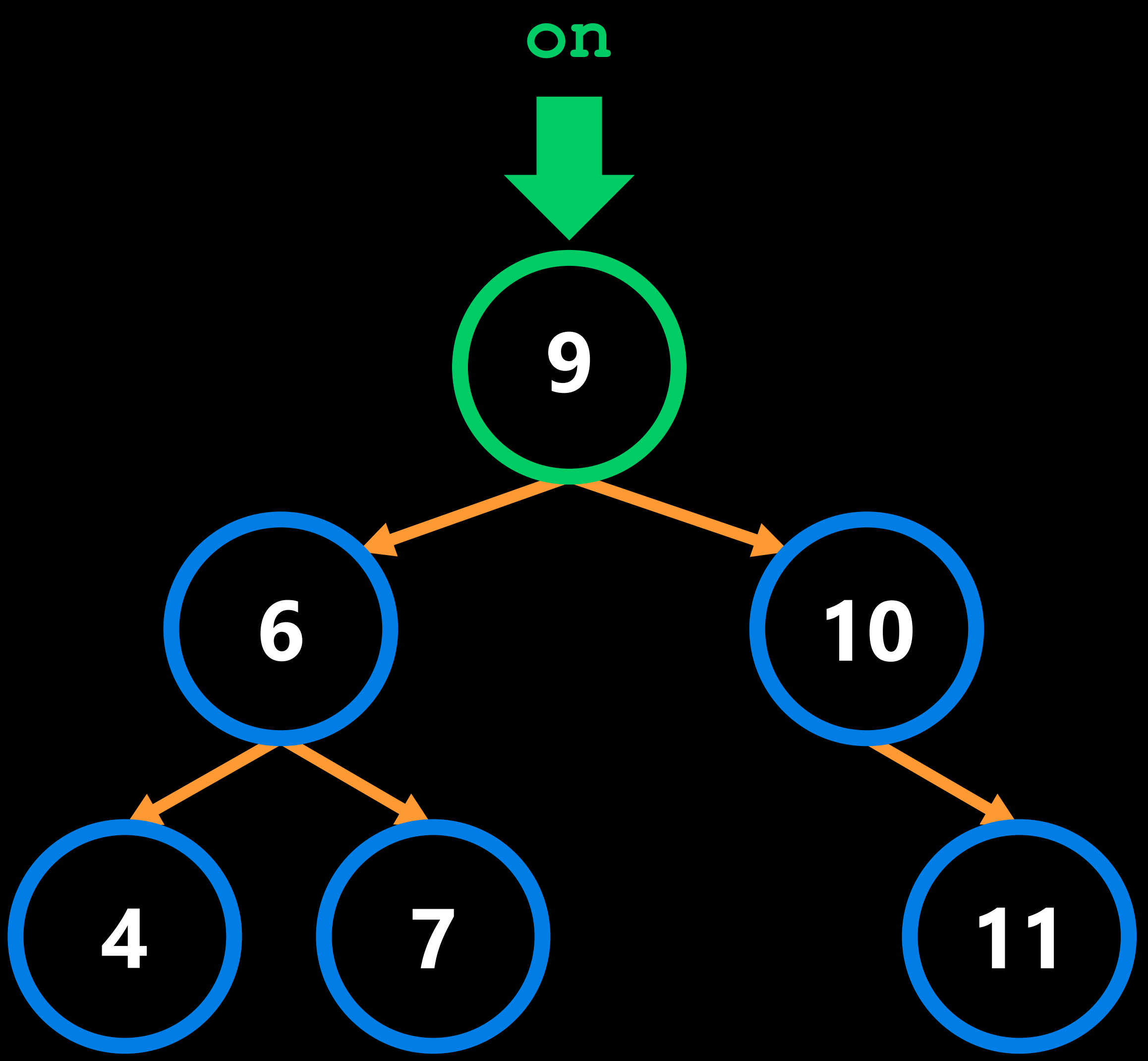
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:    True

            if cargo > on.cargo:    False  (4 < 9)
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```

tree.find(4)



tree.find(4)

```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:   ⬅ True

            if cargo > on.cargo:   ⬅ False (4 !> 9)
                on = on.right

            elif cargo < on.cargo:   ⬅ True (4 < 9)
                on = on.left

            else:
                return True

        return False
```



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:

            if cargo > on.cargo:
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```

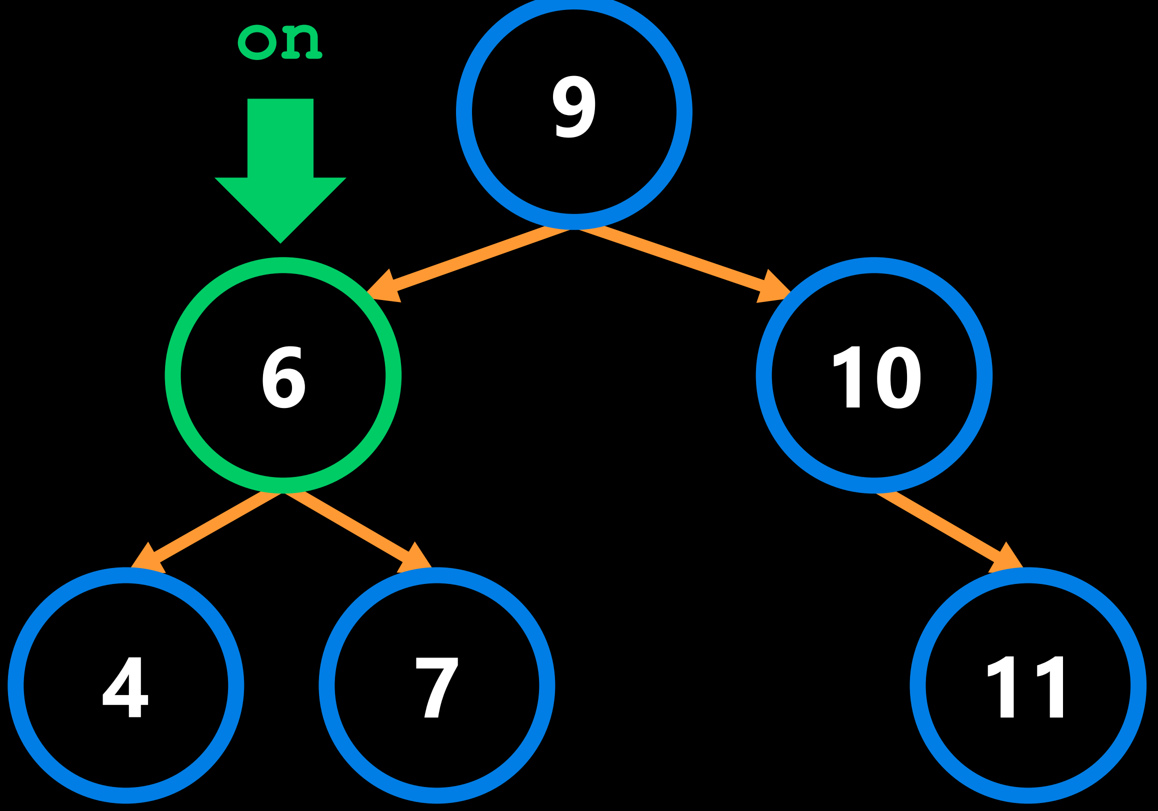
True

False (4 !> 9)

True (4 < 9)

Move on to the left.

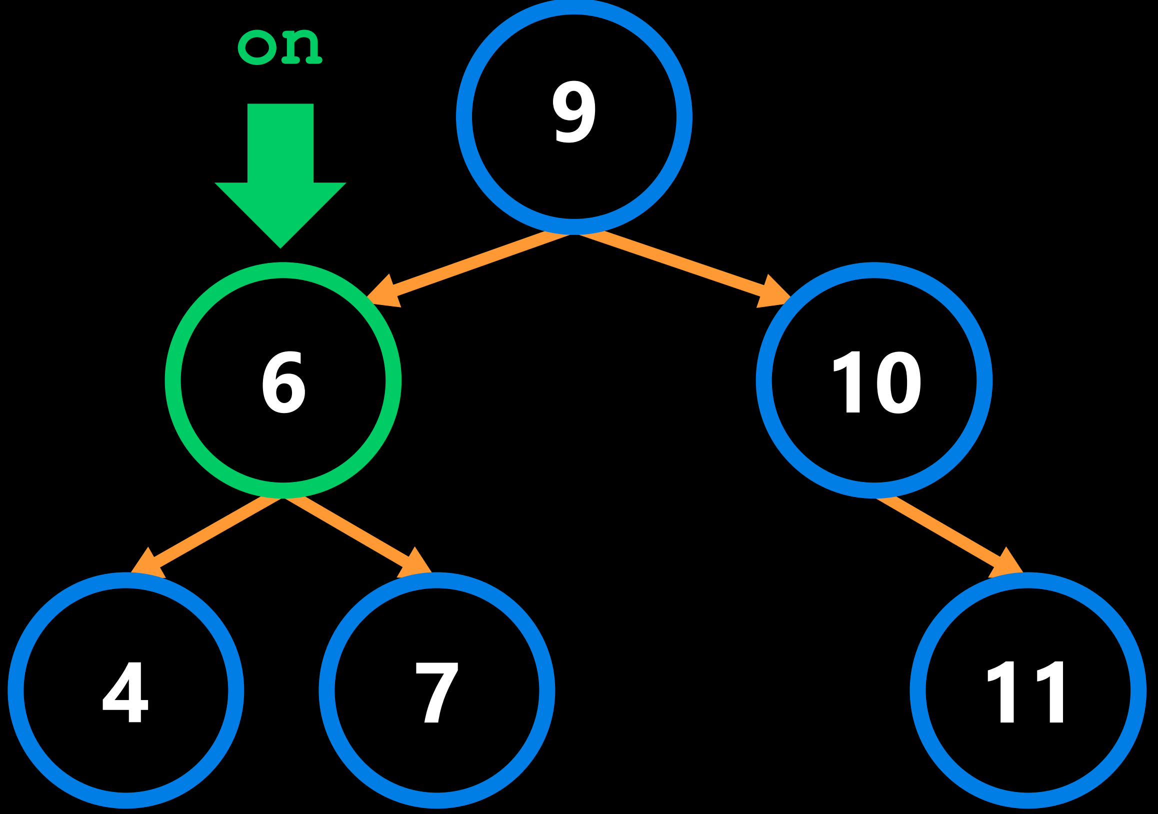
tree.find(4)



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:

            if cargo > on.cargo:
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```

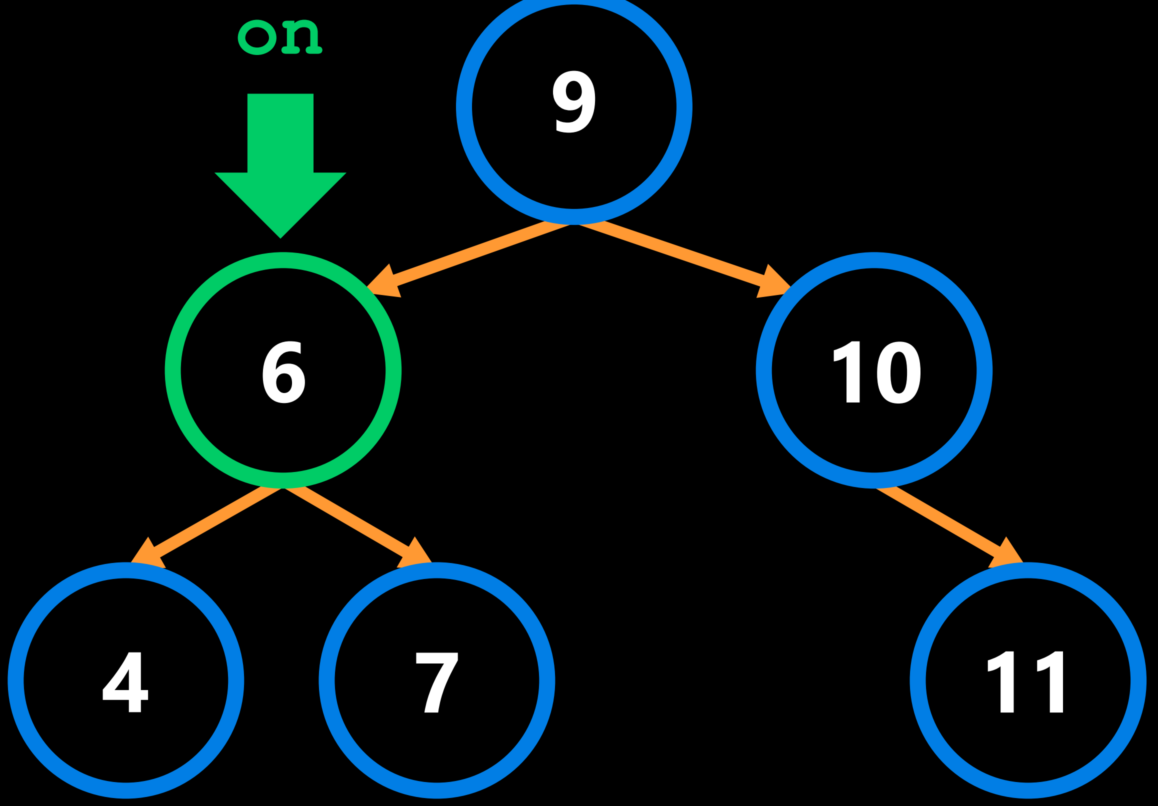
True

tree.find(4)



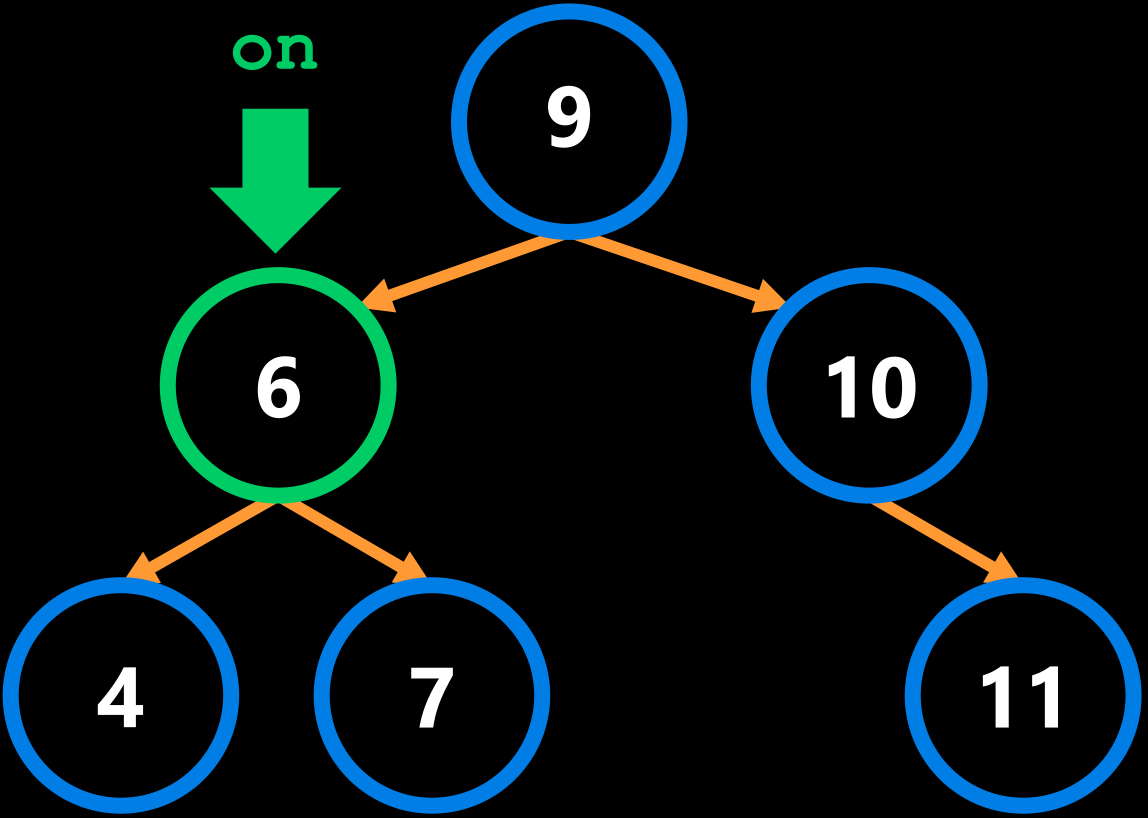
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:  ⬅ True

            if cargo > on.cargo:  ⬅ False (4 !> 6)
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```

tree.find(4)



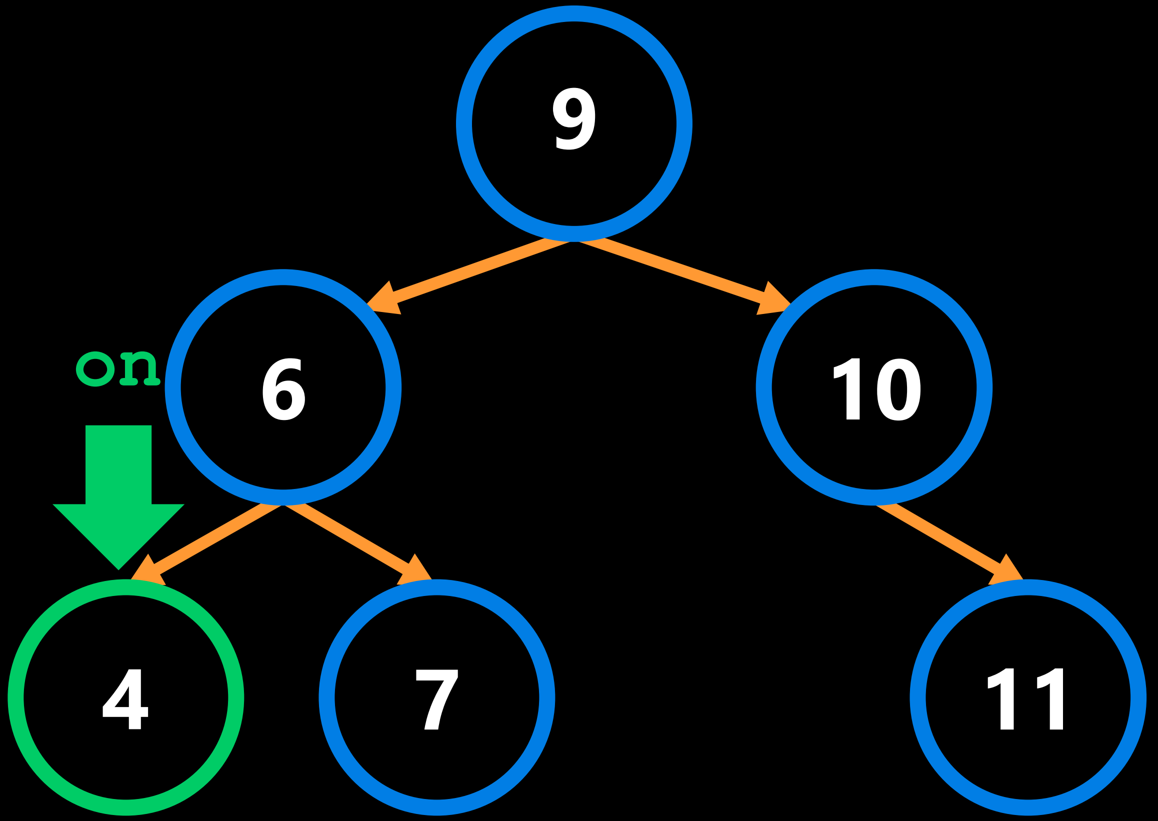
tree.find(4)

```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:  ⇐ True

            if cargo > on.cargo:  ⇐ False (4 !> 6)
                on = on.right

            elif cargo < on.cargo:  ⇐ True (4 < 6)
                on = on.left

            else:
                return True

        return False
```



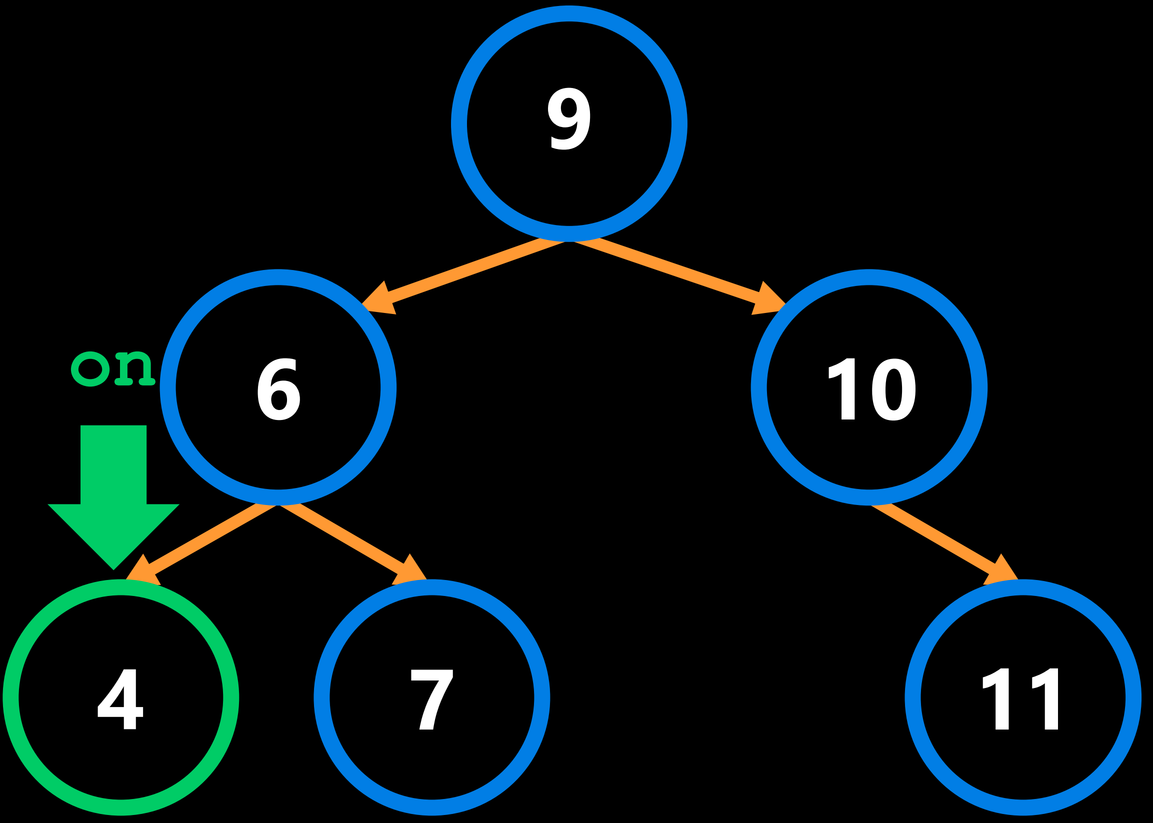
tree.find(4)

```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:        True

            if cargo > on.cargo:     False  (4 !> 6)
                on = on.right

            elif cargo < on.cargo:   True  (4 < 6)
                on = on.left         Move on to
                                     the left.
            else:
                return True

        return False
```



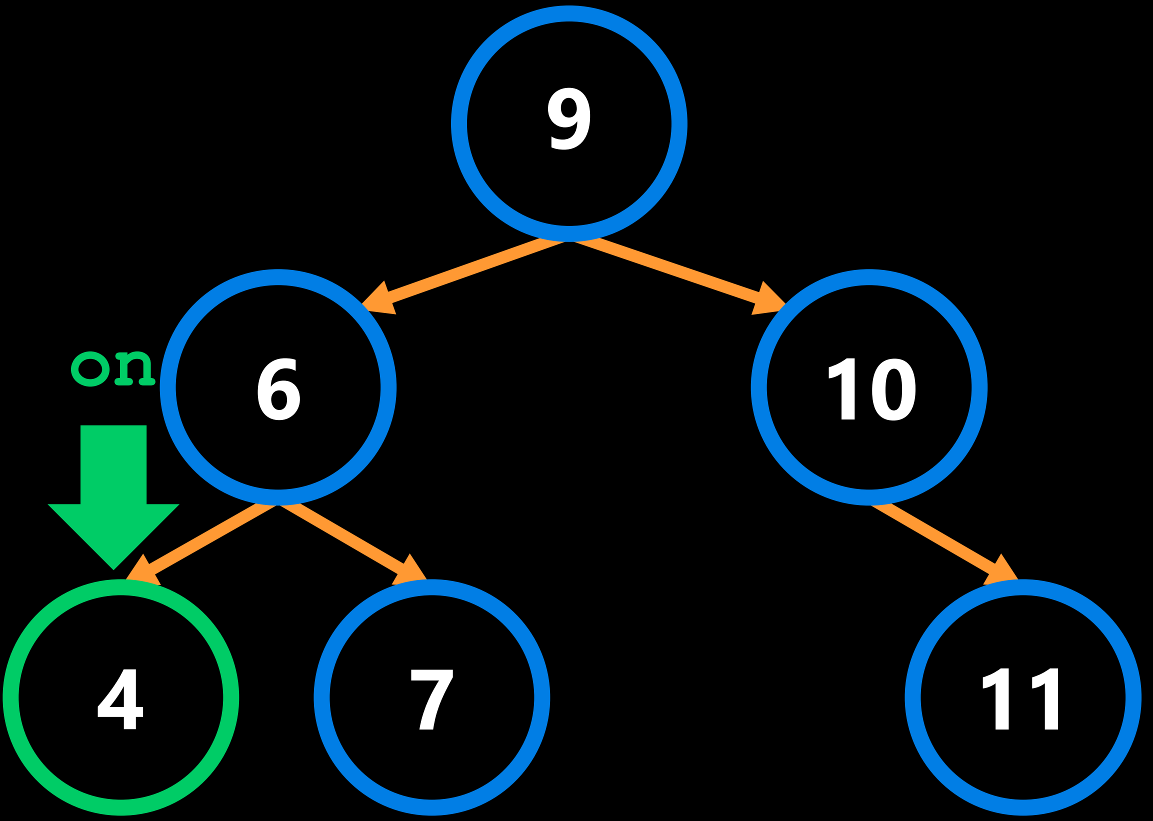
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:  ⬅ True

            if cargo > on.cargo:
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```

tree.find(4)



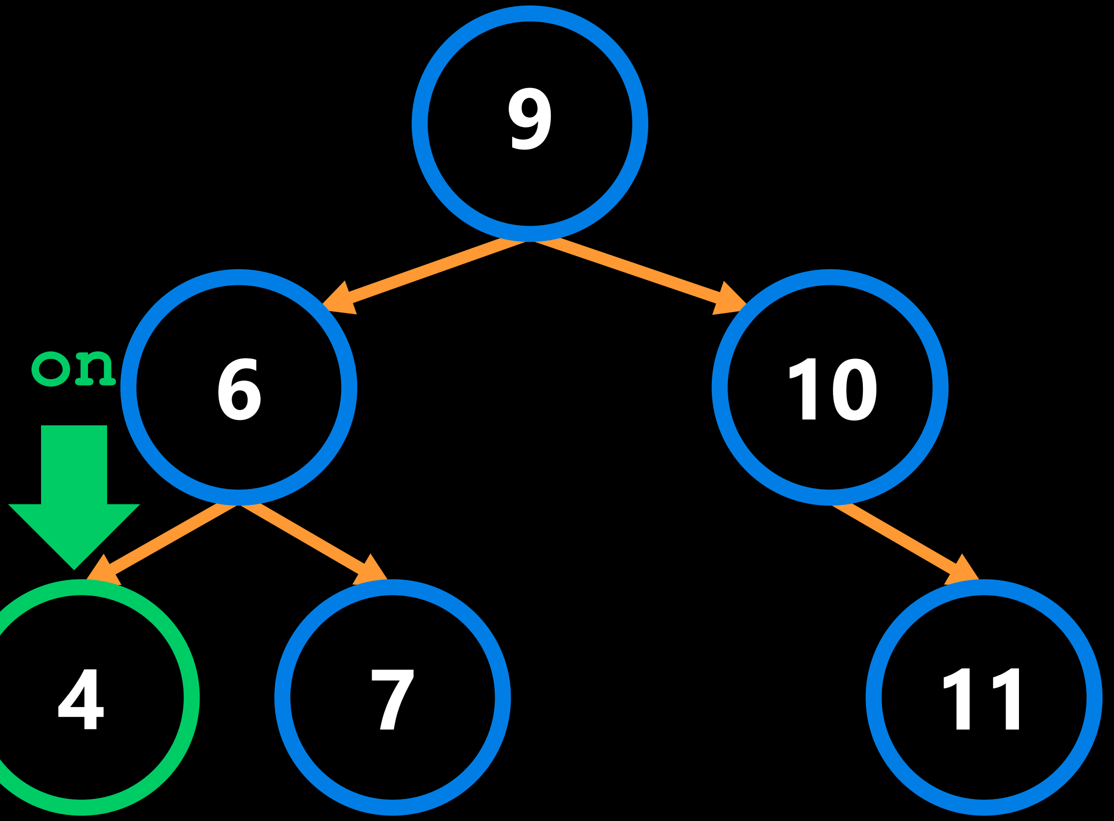
```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:  ⬅ True

            if cargo > on.cargo:  ⬅ False (4 !> 4)
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```

tree.find(4)



# tree.find(4)

```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:  ⬅ True

            if cargo > on.cargo:  ⬅ False (4 !> 4)
                on = on.right

            elif cargo < on.cargo:  ⬅ False (4 !< 4)
                on = on.left

            else:
                return True

        return False
```



```
class BinarySearchTree:
    """A Node class used by a binary search tree class."""
    def __init__(self, root=None):
        """
        (self) -> NoneType
        Create an empty binary tree.
        """
        self.root = root
        if not self.is_valid():
            print('This is not a valid binary search tree.')

    def print_tree(self): ...

    def is_valid(self): ...

    def find(self, cargo):
        """
        (self, number) -> bool
        Checks if cargo value is in the tree.
        """
        on = self.root

        while on is not None:

            if cargo > on.cargo:
                on = on.right

            elif cargo < on.cargo:
                on = on.left

            else:
                return True

        return False
```

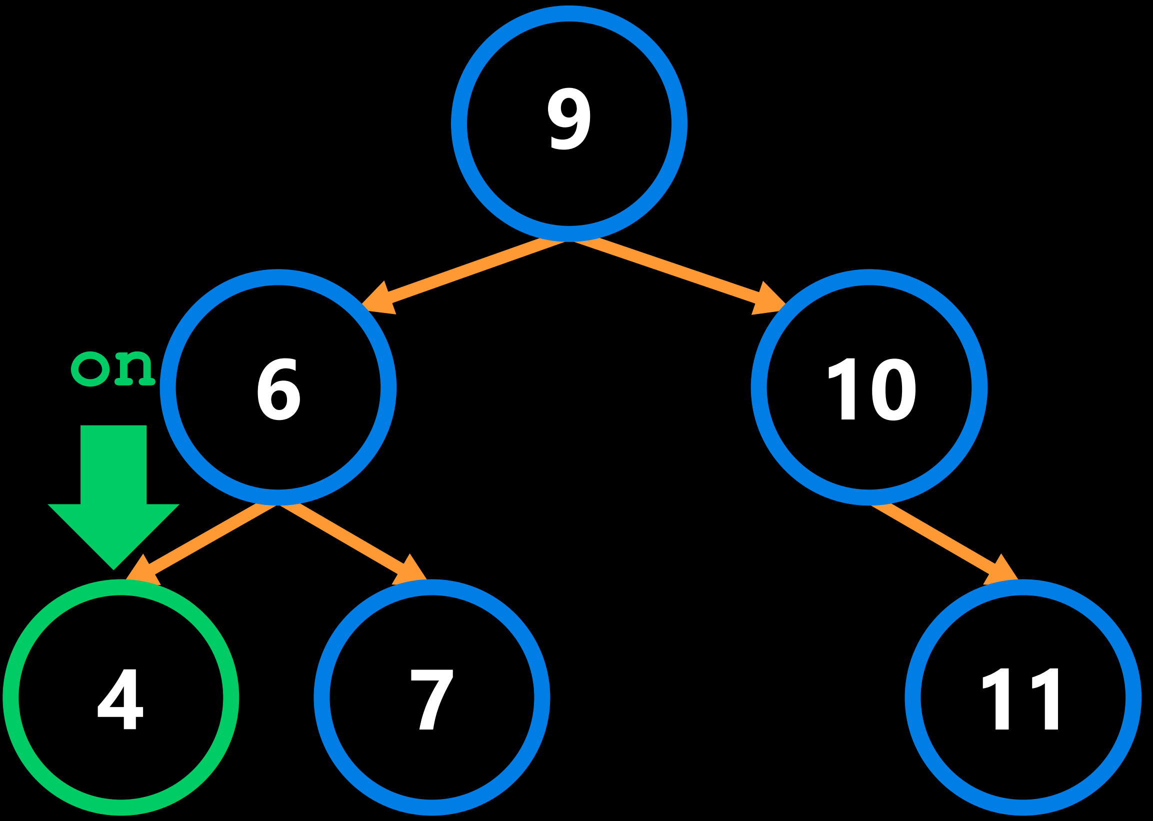
True

False (4 !> 4)

False (4 !< 4)

Value is in the tree.

tree.find(4)



# APS106

UNIVERSITY OF TORONTO

# binary search trees.

**Week 12** | Lecture 2 (12.2)

if nothing else, write #cleancode