### Data types

- None
- 3, -4, 1.01, -2.0
- True, False
- 'Hello, world!\n'
- [1, 2.0, 'hi']
- {'hi': 3, 'bye': 100}

### Basic operators

- True and False, True or False, not True
- 1 + 3, 1 - 3, 1 * 3
- 5 / 2 == 2.5, 5 // 2 == 2, 5 % 2 == 1
- 'hi' + 'bye'
- [1, 2, 3] + [4, 5, 6] # [1, 2, 3, 4, 5, 6]

### List methods

```python
def __init__(self, x, y):
    self.x = x
    self.y = y

def size(self):
    return (self.x ** 2 + self.y ** 2) ** 0.5
```

```python
p = Point(3, 4) # constructor
p.x # attribute access: returns 3
p.size() # method call: returns 5.0
```

### Exceptions

```python
raise IndexError()
```

### Class syntax

```python
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def size(self):
        return (self.x ** 2 + self.y ** 2) ** 0.5
```

```python
class MyWeirdClass(Point): # inheritance
    pass
```

### Tree class

```python
class Tree:
    # === Private Attributes ===
    # @type _root: object | None
    # The tree's root item, or None.
    # @type _subtrees: list[Tree]
    # A list of all subtrees of the tree.

    # === Representation Invariants ===
    # - If _root is None then _subtrees is empty.
    # This represents an empty Tree.
    # - _subtrees doesn't contain any empty trees

    def __init__(self, root):
        #"""Initialize a new Tree with the given root value."
        # If <root> is None, the tree is empty.
        
        @type self: Tree
        @type root: object | None
        @rtype: None
        """

    def is_empty(self):
        #"""Return True if this tree is empty.
        
        @type self: Tree
        @type: bool
        """
```

### BinarySearchTree class

```python
class BinarySearchTree:
    # === Private Attributes ===
    # @type _root: object | None
    # The BST's root value, or None.
    # @type _left: BinarySearchTree | None
    # The left subtree, or None if the tree is empty.
    # @type _right: BinarySearchTree | None
    # The right subtree, or None if the tree is empty.

    # === Representation Invariants ===
    # - If _root is None, then so are _left and _right.
    # This represents an empty BST.
    # - If _root is not None, then _left, _right are BSTs.
    # - Every item in _left is <= _root, and
    #   every item in _right is >= _root

    def __init__(self, root):
        #"""Initialize a new BST with a given root value."
        # If <root> is None, the BST is empty.
        
        @type self: BinarySearchTree
        @type root: object | None
        @rtype: None
        """

    def is_empty(self):
        #"""Return whether this tree is empty.
        
        @type self: BinarySearchTree
        @type: bool
        """