Jiaxuan Pan (jp4131)
Qianjun Chen (qc2300)
Eurey Noguchi (yn2377)
Roger Lu (jl5822)

JQER - PLT Proposal

1. Overview

"JQER" is a Python-like language for binary-tree data structures and operations on them. The main goal of JQER is to simplify the operations on binary-tree data structures. The language integrates Node and Tree as built-in data types and their associated operating modules.The syntax of the language is similar to Python and is designed as a dynamically-typed objects definition. However, the language introduces additional rules which are applied to the tree data types.

2. Language Manual

2.1 Data Types

The built-in data types are integers, floats, booleans, strings, nodes, and trees and the language will be dynamically and strongly typed.

Data type	Notation	Description	Initialization
Numeric	int	Positive or negative whole numbers. It has a size of 4 bytes.	a = 1
	float	Real numbers written with a decimal point separating the integer and the fractional parts. It has a size of 8 bytes.	a = 1.5
Boolean	boolean	Value is either true or false.	a = true
Text	str	Immutable sequences of Unicode code points.	a = "hello"
Tree	tree	Store the address of the root Node. It can construct a tree from a single node or multiple nodes in which the subsequent nodes will be added to the most right pointer of the previous	tree1 = Tree([node1,node2…]) tree2 = Tree(node1)

		node.	

Node	node	Stores the value, left pointer, right pointer.
		Only nodes with the same data type can be in a tree
			node1 = Node(2.5)
			node1 = Node(“hi”)

2.2 Operators

The language is implementing operators =, ==, !=, +, -, *, /, +=, -=, <, >, >=, <=, and, or, not. The operators are applied to most of the data types we defined above but not always support for every one.

2.2.1 Arithmetic Operators

Operation	Data Type (Notation)	Description	Example
+	Int, float, str	Addition Specials: float + int = int str + int/float = error	1 + 2 = 3 1.5 + 2.2 = 3.7 1.7 + 3 = 4 “Hel” + “lo” = “hello”
	tree, node	tree can be added to another tree at the most-right pointer. node can be added to a tree at the most-right pointer.	tree1 + tree2 = tree1 tree1 + node1 = tree1
-	Int, float	Subtraction Specials: float - int = float int - float = int	2.5 - 1 = 1 2 - 1.5 = 1
	tree, node	Option1: tree can remove a node and its linked nodes. Option2: tree can remove a node which node.value is equal to.	Option1: tree1 - node1 = tree2 which is without all nodes below(linked to) its. Option2: Tree1 - 2 = Remove nodes which node.value == 2
*, /	Int, float	Multiplication, Division	2 * 2.3 = 4 3 / 2.5 = 1

2.2.2 Assignment Operators

Operation	Data Type (Notation)	Description	Example

=	Int, float, str, boolean	Assign value to variable	int1 = 2
			boolean1 = true

	tree, node	Same as above	tree1 = Tree()
			node1 = Node(2)
			tree2 = Tree(node1)

+= -=	Int, float	Equal to add the value to the variable	int1 += 3.4

	tree, node	Equal to add the value to the variable	tree1 -= node1
			tree1 += node1

	str	Only support +=	a = “hell”
			b = “o”
			a += b

2.2.3 Comparison Operators

Operation	Data Type (Notation)	Description	Example
>, >=, <, <=	int, float	larger/less (or equal) than. int is converted to float in comparison with float.	4 > 3 return true 1 <= 1.5 return true
== !=	int, float, str, boolean	Check if the value is the same, but not the address. int is converted to float in comparison with float.	“Hello” != “hello!~” return true 1 == 1.0 return true
	tree, node	Check if the address and the size of all nodes are the same.	tree1 == tree1 return true node1 != node1 return false tree1 == node1 return Err

2.2.4 Logical Operators

Operation	Description	Example
and	Returns True if both statements are true	x<5 and x>3
or	Returns True if one of the statements is true	x<5 or x>3
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

2.3 Syntax

Syntax will be close to the syntax of python. All the rules for original functionality will remain the same. Some additional rules for typing and new functions for trees will be added. To support Object-Oriented Programming, the language can define basic classes, but without any inheritance mechanism. Classes are simply objects with associated methods, local variables, and constructors.

Description	General	examples
Assigning value	var = value1	i = 1

Defining Function	def func(arg1, arg2, ...):
	    //code
		def printSize(tree):
		    print(tree.size())

Defining Class	class ClassName:
	    var1 = value1
	
	class ClassName:
	    def __init__(self, value1, value2):
	        self.var1 = value1
	        self.var2 = value2
		class MyClass:
		    node = 5
		
		class Student:
		    def __init__(self, name,grade):
		        self.name = name
		        self.grade = grade
		
		    def printName(self):
		        print(self.name)
		
		s1 = Student(“abc”, 1)
		s1.printName()

Calling Methods	var.func(arg1, arg2, …)	arr.append(1)
		t1.balance()

	func(arg1, arg2, ...)	print(var1)

Loop	for item in list_or_range:
	    //code
	
	while condition:
	    //code
		for i in range(5):
		    if i == :
		        break
		    if i == 2:
		        continue
		    print(i)
		
		while i < 6:
		    if arr[i] == 3:
		        break
		    print(arr[i])
		    i += 1

If Condition	if condition:
	    //code
	elif condition:
	    //code
	else:
	    //code
		if i == 0:
		    print(i)
		elif isIncrement is True:
		    print(i+1)
		else:
		    print(0)

2.4 Functions

2.4.1 Standard Library

- A list data structure with append, insert, remove, pop, len methods is built in as a standard library.
- range(int) which takes an integer and creates an immutable sequence type.
- print(obj) which prints objects to stdout.
- type(obj) return the type of the object
- to_bst(tree) which takes a tree and turns it into a binary search tree.
- balance(tree) which takes a tree and makes it balanced.
- contains(tree, int/str/float) which takes a tree and checks if any node's value meet the given value.
- depth(tree) which returns the tree's depth.
- inorder(tree, func) takes a tree and a function and traverses each node inorder applying the function.
- preorder(tree, func) takes a tree and a function and traverses each node preorder applying the function.
- postorder(tree, func) takes a tree and a function and traverses each node postorder applying the function.

2.4.2 Function Declaration

Function is declared using python syntax, where it either has or does not have a return value. Function supports recursion and functions could be declared inside a function.

```
def fib_recur(n):
    if n <= 1:
        return n
    return fib_recur(n-1) + fib_recur(n-2)
```

2.4.3 Comments

Comments are designed to use python syntax, with a pound key to indicate the start of a single line comment and three pairs of closed single quotation marks to indicate a multi line comment.

```
# This is a single line comment.

'''
This is a multi-line comment
That has two lines
'''
```

3. Sample Code

3.1 Making Balanced Binary Search Tree

```
tree = Tree([1, 2, 3, 4, 8, 5])
bst = to_bst(tree)
balanced_bst = balance(bst)
def print_tree(x):
    print(x)
inorder(balanced_bst, print_tree)
```

3.2 Hello World!

```
def helloWorld():
    node1 = Node(1)
    node2 = Node(2)
    tree1 = Tree(node1)
    tree2 = Tree([node1, node2])
    if tree1 != tree2:
        print(“Hello World!”)
    else:
        print(“Good-bye World!”)
helloWorld()
```