ROLE PLAY

Functions-Stack-Variable scope

GRADES 7 AND ABOVE

Have been introduced to functions and procedures

Program and sub programs

- Mini Calculator program
- Scenarios
  - A programming language, that does not allow variables being shared between procedures
  - Function: Does not return a value
  - Procedure: Returns a value

Purpose

- Purpose of passing parameters.
- Can you access local variables of one block in another?
- Do you ideally need parameter passing in this scenario in Scratch?
- What if we have another program where one function, is called from another?
- OR Where functions are there, no return value?
- OR Where procedures are there with return value?
- Compare languages

Probe points

- First sub program that executes?

Role play

- Player 1: Main sub program
- Player 2: Initialize
- Player 3: Calculate with +
- Player 4: Display

The pencil box (a cylindrical, standing one) is the stack. Drop in post-it for the variables, parameters and return values, whenever necessary and each kid gets a role by taking up a procedure.

How to play?

STACK: Like a stack of books. The one you put first, goes to the bottom. Called LIFO: last in first out. Procedure calls and variables in procedures (including the main procedure) is implemented in code by the programming language's compiler or interpreter using a stack **abstract** data type.

We saw the list **abstract (**not real, in theory, can't see, a layer**)** data type, if we block one head of the list

- Create post-its for variables and parameters
- Parameters: Go into the stack
- Local variables go into the stack
- Return location goes into the stack
- Stack addresses reduce.

Scratch program

- Refer to Add_Diff_Mul_Div

Parameter passing: Need

- Variables not visible in sub programs, then need to pass

Call Stack

Initialize call	Accept C all	Calculate	Display
		Operator	
		P:num2	Local: “2”
		P:num1	Local: “Sum:”
	Local: Answer	Calculate	Display
Initialize	Accept Input	Local: operator	Local: operator
sum	Sum	Sum	Sum
num2	num2	num2	Num2
num1	num1	num1	Num1
Main stack	Main	Main stack	Main

Program: Variables not shared between sub programs: as in python

- Main()
- Declare num1, num2, sum
- list(num1, num2, sum) Initialize()
- Initialize num1 and num2 and sum with the values returned
- List(num1, num2) AcceptInput()
- set num1 and num2
- Sum = Calculate(num1, num2) (return value of calculate is assigned to sum)
- Void Display(sum)

Code

Refer to Calculator.py

Refer to https://www.cs.ucsb.edu/~pconrad/cs8/topics.beta/theStack/05/

Call Stack: Variables not shared

- Refer to “Call_Stack_Game.docx”

Try

- Reform the stack to incorporate the print/join call (Any call to a built – in function).
- Recursion: The function call in which the same function is called again
  - F()
  - {
    - F()
  - }

Global scope

- Try changing the num1, num2 and add to “global” variables.
- Need not pass it on between sub programs
- Will become similar to the “Scratch” program

The heap and stack

- Create a class “Calculate”
- Member variables: num1, num2 and sum
- Member functions: initialize (explain constructor for older kids), acceptInput, calculate, display

Heap

- (Easier to explain for kids who know bits and bytes and space occupied)
- Size of a string in Scratch
- Does Scratch know beforehand, how much space it would take?

- Think in case of an integer. Integers have an upper limit and Scratch knows till that number, integer will take only so much space

- Similarly, the language like JAVA or Python does not know in advance what is the size of the class, since it does not know, how big you plan to make it, in terms of adding member variables
- Such data types are stored in a separate area in memory called the **HEAP**

Calculate using heap

- Instantiate the Calculate class
- Means: Memory is allocated for the class on the heap
- Draw a square on the ground to depict it
- Add the Calculate object on to the stack
- Num1, num2 and sum are on the heap
- The function calls are on the heap

Pseudo Code

- Class
- {
- Main()
- {
  - Calculate obj = new Calculate();
  - Call AcceptInput() in object
  - Call calculate() in obj
  - Call display() in obj
- } **obj reference popped off here**
- }

Heap and garbage collection

- Works similarly.
- Once the functions are all popped, obj reference is popped.
- The object on the heap is marked for removal from heap by the garbage collector in java and should be deleted by the programmer in C++

- Heap should be managed by the programmer
- Stack is managed by the programming language

- For advanced kids:
  - Calculate obj = new Calculate();
  - Calculate obj = new Calcuate();

  What happens to object created on the heap in the first line?

  Ans: Garbage collected in Java. C++, take care to delete it.

HOW WAS THE RIDE?

Thanks