3) What programming paradigm started taking over the world in the 1980s?

Answer :: Object-orientationstarted taking over the world in the 1980s

Though begun in the 1960s with Simula and refined in the 1970s at Xerox PARC with Smalltalk, OO—or approximations to it—explodes in the 1980s with C with Classes (since renamed C++), Objective-C, Eiffel, and Self. Earlier languages such as Lisp and Pascal gain OO features, becoming Common Lisp and Object Pascal, respectively.

4) In Bret Victor’s *The Future of Programming*, what are the four “predictions” his 1973 persona predicted for 2013 that never really took hold?

Answer :: (1) coding direct manipulation of data

(2) procedures goals and constrains

(3) text dump spatial representations

(4) sequential concurrent

11) What is the difference between imperative and declarative programming?

Answer :: **Declarative programming** is a programming paradigm that expresses the logic of a computation without describing its control flow.

**Imperative programming** is a programming paradigm that uses statements that change a program’s state

15) Syntax deals with \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and semantics deals with \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

Answer :: Syntax deals with structure and semantics deals with meaning

20) What is the difference between concrete syntax and abstract syntax?

Answer :: concrete syntax is define by a context free grammar

* It consists of a set of rules ( production ) that define the way programs look like to the programmer

Abstract syntax is the set of trees used to represent program in the implementation

* The abstract syntax define the way the programs look like to the evaluator/compiler

24) What are denotational semantics, operational semantics, and axiomatic semantics?

Answer :: **Denotational semantics**  ties identifiers to their meaning . It's when you define a function it should do what it says.

Example , if you define a function like says add\_numbers(x, y) it should add x to y and not multiply them

**Operational semantics**  ties any type of operation like arithmetic, assignment to the computation.

Two types of operational semantics

1. structural operational semantics = describe how the individual steps of a [computation](https://en.wikipedia.org/wiki/Computation) take place in a computer-based system.
2. natural semantics = describe how the overall results of the executions are obtained

**Axiomatic semantics**  define the meaning of a command in a program by describing its effect on assertions about the program state

The assertions are logical statements like Boolean algebra , code branching (if-elif-else, switch-cases) , predicates with variables where

the variables define the state of the program.

28) What is an assignable? How does assignment differ from binding?

Answer :: assignable means sets and/or re-sets the value stored in the storage location by using  variable name like it copies a value into the variable

assignment can only change the value of a given variable within the narrowest bound scope while Binding is used to create a new variable within the current context

34 ) How does Python prevent a binding from outliving the entity it is bound to? How does Rust prevent this from occurring?

Answer :: x=3 //value is 3, bind x

def f(): x //f return value of entity bound to x

x=5 //replace value of entity

f() //5

40) What would this program output under (a) static scope rules and (b) under dynamic scope rules?

var x = 2;  
function f() { print x; }  
function g() { var x = 5; f(); print x; }  
g();  
print x;

Answer :: 2 (first call function f . it has x =2 value)

5 (call function g. it has x = 5 value)

2 (call x variable. it has x =2 value)

45) What is the difference between the way JavaScript and Rust handle the sequence

Answer :: JavaScript = let x = 3; // bind value 3 to x

console.log(x); // print 3

x = 8; // reassign value 8 to x

console.log(x); // print 8

Rust = let x = 3; // bind value 3 to x

println!("{}",x); // print 2

x =8 ; // error because you cannot reassign value

println!("{}",x);

* If we want to change value in existing variable then must use **MUT**

let mut x = 3; // bind value 3 to x

println!("{}",x); // print 2

x =8 ; // reassign value 8 to x

println!("{}",x); // print 8

49) What is operator precedence? What does is mean for operator O1 to have higherprecedence than operator O2? Give a precise example

Answer :: When several operations occur in an expression, each part is evaluated and resolved in a predetermined order called operator precedence

Example, So operator O1 is addition and operator O2 is multiplication. So , have higherprecedence than operator O2 because of parenthesis

console.log((2+10)\* 2); //24

console.log(2+(10\* 2)); //22

console.log(2+10\* 2); //22

51) Evaluate, if possible, the expression in -2\*\*2 in JavaScript and Python. Explain why the evaluation produced the value it did in each language.

It is possible in python.

Not possible in javascript1

52) Why would a language define an evaluation order for expressions? Why would it choose to leave the evaluation order undefined?

Answer :: language define an evaluation order for expressions because it used to compute and to assign values to variables and to help control the execution flow of a program.

Requirement must be met for each allowable ordering of the subexpressions of a full expression otherwise, the behavior is choose to leave the evaluation order undefined.

56) What does the following script output under lazy evaluation? Under eager evaluation?

let x = 5  
function f() { x = x \* 3 }  
function g() { x = x \* 5 }  
function h(a, b) { return a + x }  
print(h(f(), g()))

Answer :: lazy evaluation 30

Eager evaluation 90

59) How did Go “fix” the most unintuitive aspect of the C (and inherited by C++, JavaScript, Java) switch-statement?

Answer :: Go only runs selected cases , not all the cases that follow.

‘Break’ statement is required at the end of each case.

There is no need to b constants in Switch cases no need to b values as integers.

Except this, Go switch case is same as c , c++ , java , javascript.

60) Why is it that the simple act of doing an operation 10 times so easy in Ruby but so annoyingly complex in C-like languages?

Answer :: In Ruby,

* + - * there’s no macros or pre-processor, no casts, no pointers, no typedefs, sizeof, nor enums available.
      * All variables live on the heap. Further, you don’t need to free them yourself—the garbage collector takes care of that.

In C

* they are present
* we need to free them our self because garbage collector is not present in C

62) What is the non-deterministic statement of Go? Of Erlang?

Answer :: select is the non-deterministic statement of Go

Case is the non-deterministic statement of Erlang.

66) What are the main differences between a type and a class?

Answer :: **Type**  refers to the classification of primitive values such as integers, strings, arrays, Boolean, null, etc. We cannot create a new one

**Class** refers to the named set of properties and methods which an object is associated with when it is created. We can create as many new classes as we want.

68) Why to languages like Ruby have a single type Array but in other languages like Java, Rust, and Swift there exist many parameterized array types?

Answer :: In Ruby. arrays can contain any datatype, including numbers, strings, and other Ruby objects so ruby have a single type Array.

72) What are three primary situations in which a type A is compatible with type B?

Answer :: (1) if the two types are same

(2) if the right side type is a subtype of the left side

(3) if the type of the right side can be converted to the type of the left side.

77) What is the difference between strong typing and weak typing?

Answer :: weak typing In a weakly typed language, the type of a value depends on how it is used.

* + - * + weak typing is delaying checking the types of the system as late as possible, usually to run-time

strong typing a value has a type and that type cannot change. What you can do to a value depends on the type of the value.

* + - * + checks the type of variables as soon as possible, usually at compile time

78) What is the difference between static typing and dynamic typing?

static typing Static typed programming languages are those in which variables need not be defined before they’re used.

Example JAVA

statically typed languages perform type checking at compile time

dynamic typing Dynamic typed programming languages are those languages in which variables must necessarily be defined before they are used

Example Python , PHP

dynamically-typed languages perform type checking at runtime

80) Why is the combination of static typing plus weak typing the worst of the four possible combinations?

Answer :: weak typing allows implicit conversations and making the code less explicit with readable , which is a major benefit of static typing . Unfortunately, static plus weak typing removes the sureness that static typing provides.

85) Given the types A = {1, 2, 3} and B = {'a', 'b'}, what are the types A+B and A×B?

Answer :: tuples

86) Untagged product types are usually called \_\_ structures \_\_. Tagged product types are usually called \_\_\_sumtype\_\_\_. (There are two good answers for the latter; one starting with “s” and one starting with “s”.

90) In what way is a struct like a dictionary? In what why are they different?

Answer ::

Student = struct(…

‘name’,{‘jon’,’khaleesi’,’arya’},

‘subject’,{‘maths’,’history’,’science’},

‘grade’,{‘95’,’88’,’78’});

structs are for structured collections of data, while dictionaries are for long lists of named elements

94) If someone talks about static arrays and dynamic arrays, what are they probably referring to?

Answer :: If someone talks about static arrays then they referring to array with **fixed** in size and **Size** of static arrays should be**determined at compile-time.**

If someone talks about dynamic arrays then they referring to array with not **fixed** in size and allocated on **heap.** **Size** of dynamic arrays can be **determined either at compilation or at run-time**

96) How do Go and Rust differ (if at all) in their interpretation of the length of a string?

Answer :: To get the length of a String in Go, use the function len().It gives character

To get the length of a String in Rust, use chars() and count() because len() gives byte, not character.

102) What is the difference between a subroutine and a coroutine?

Answer :: subroutine invoked once and executes until it completed

can be translated to a coroutine which does not call yield

coroutine Coroutines is a more generalized form of subroutines.

Can be entered, exited, and resumed at many different points.

Can pause execution and yield control back to the caller or another coroutine.

103) What is the difference between a parameter and an argument?

Answer :: A parameter is a variable in a method definition. When a method is called, the arguments are the data you pass into the method's parameters

Example :: height and weight would be referred to as its parameters

int add(int height, int weight) {

return height + weight;

}

Example :: 50 and 60 would be referred to as its argument

int result = add(50, 60);

107) What is a rest parameter?

Answer :: A function can be called with any number of arguments, no matter how it is defined.

Example :: let str =”hello”

Alert([…str]);

110) Explain the following five pragmatic mechanisms for argument passing (value, value-result, reference, name, pure-aliasing)

Answer :: value copy going into the procedure. this mechanism is used for passing objects, where a reference to the objected is passed by value.

Advantages :: Argument protected from changes in callee

Disadvantages :: Copying of values takes execution time and space, especially for aggregate values

value-result copy going in, and again going out

Advantage :: is more efficient than copying

Disadvantage :: can redefine constants

Reference The actual parameters and formal parameters are identified. The natural mechanism for this is to pass a pointer to the actual parameter, and

indirect through the pointer

Disadvantage :: if an error occurs, harder to trace values since some side-effected values are in environment of the caller

Name re-evaluate the actual parameter on every use. For actual parameters that are simple variables, this is the same as call by reference. For actual parameters that are expressions, the expression is re-evaluated on each access. It should be a runtime error to assign into a formal parameter passed by name, if the actual parameter is an expression.

Implementation :: use anonymous function ("thunk") for call by name expressions

pure-aliasing argument and parameter are one. Changes to parameter immediately are seen in the argument, because, as was just mentioned,

the argument and parameter are one.

112) What is a closure?

Answer :: . A closure is a subroutine that refers to variables defined in an enclosing scope.

117) Alan Kay said, in a letter to Stefan Ram, “OOP to me means only \_\_\_ messaging\_\_\_, local retention and protection and hiding of \_\_\_ state-process\_\_\_, and extreme \_\_ Late Binding of all things\_\_\_.”

118) The two major approaches to OOP are characterized by \_\_object\_\_ and \_\_dynamic binding\_\_. Which of the two is more associated with Plato and why?

Answer :: object and dynamic binding are major approaches in OOP

Object and inheriting are more associated with Plato because,

Object is describing what the **object** will actually be like and what **properties** it will have when created

i**nheriting** all the properties of the parent class, but enhancing it with additional properties to create a new “child” class containing all of its parents’ properties

119) What might have caused the proliferation of getter and setter methods in languages like Java and C# that purport to be object oriented? What are the arguments as to why getters and setters are evil?

Answer :: getter and setter methods are proliferating because  these methods are in a lot of framework and used as a reflection mechanism.

getter and setter methods are evil because

* They are a waste of CPU time and programmer time. Not just for the people writing the code and tests, but for the people who have to read and understand them as well.
* If we can get an object out of another object, we are relying too much on the first object’s implementation details. For instance object will change then, we have to change our code as well.

122) Explain the difference between the Erlang-style and the Go-style of process communication.

Answer :: GO Goroutines in Go, lightweight threads of execution independent from the system’s thread

* Functions can be created and attached to struct types, but the style is more composable to facilitate the growth of projects over time. Rather than embedding the functions within an Object that must be expanded, the function can be created anywhere and attached to the type.
* Variables are immutable, and message passing is used so no pointers are passed around, meaning that function calls are very literal in their operation
* the patterns are defined outside of a function for reuse but can result in creating a lot of duplicate interfaces if they aren’t well organized

ERLANG Erlang’s actor model in which the lightweight threads are the main entity which is directly addressable

* Erlang favors a more functional style but blends some principles from object-oriented languages that make the transition seem less foreign.
* Due to immutable data, common operations such as for-loops aren’t available because incrementing a counter isn’t an option
* can’t reuse the patterns as easily, but the pattern is always defined in the exact place where it is used.

**Syntax and Semantics section**

5) (a = 3) >= m >= ! & 4 \* ~ 6 || y %= 7 ^ 6 & p

**Names**

4) Show the output of the following, assuming dynamic scope and (a) deep binding, and (b) shallow binding

function f(a) {

let x = a - 1

function g() {

print x - 17

}

h(g)

}

function h(p) {

let x = 13

p()

}

f(18)

Answer :: 0

-4

### Pointers and References

1) If possible, write a program in Go that makes a variable point to itself. That is, for some variable x, make it so that \*x == x. If this is not possible, state why it is not possible

Answer :: it is not point it self

* When you declare a pointer variable, you follow the variable name with the type of the variable that the pointer points to, prefixed with an \*, like this:

var x \*int32 = &someint

* Where, & = address, or a pointer to that variable.

### Control Flow

### The following pseudocode shows a midtest loop exit

1. while (true)  
       line := readLine();  
       if isAllBlanks(line) then exit end;  
       consumeLine(line);  
   end;

### Show how you might accomplish the same task using a while or repeat loop, if midtest loops were not available. (Hint: one alternative duplicates part of the code; another introduces a Boolean flag variable.) How do these alternatives compare to the midtest version?

### Answer :: Line = readLine();

### While (!IsAllBlanks(line))

### {

### consumeLine(line);

### }

### Line = readLine(); //call again readline()

### Subroutines

### 14) Explain what is printed under (a) call by value, (b) call by value-result, (c) call by reference, (d) call by name.

x = 1; y = 2;

function f(a, b) {

a = 3; print b, x;

}

f(x, x + y);

print x;

Answer :: A 3,1,1

B 3,1,3

C 3,3,3

D 5,3,3

JavaScript reinforcement problems

73) Given let f = async x => 1 and let g = async => 2 what do the expressions f() and g() return? Why? Why is the definition of g even acceptable?

Answer :: function f return promise 1

Function g return 2 as async as parameter