Which of the following programming paradigms allow us to write programs and know they are correct before running them?

* 1. Automata based Programming Paradigm
  2. Logical Programming Paradigm
  3. **Dependent type Programming Paradigm**
  4. Imperative Programming Paradigm

1. Which of the following is false regarding dependent types?
   1. They allow us to write programs and know they are correct before running them.
   2. **They allow us to write programs and know they are correct only after running them**.
   3. You can specify types that can check the value of your variables at compile time.
   4. Its definition depends on a value.
2. Which of the notations is true for “P(x) is true for all values of x in the universe of discourse”?
   1. x∀P(x)
   2. ∀P(x)x
   3. ∃xP(x)
   4. **∀xP(x)**
3. Which of the following is false about quantifiers?
   1. Notation ∀ is used for Universal quantifier.
   2. Notation ∃is used for Existential quantifier.
   3. “All men drink tea” is an example of Universal Quantifier
   4. **“All men drink coffee” is an example of Existential Quantifier.**
4. Let P(x) be the predicate “x must take a discrete mathematics course” and let Q(x) be the predicate “x is a computer science student”.

Which of the following statements is correct for “Everybody must take a discrete mathematics course or be a computer science student”?

* 1. **∀x(Q(x) V P(x))**
  2. ∀x(Q(x)) V ∀x(P(x))
  3. ∀x(Q(x) || P(x))
  4. ∀x(Q(x) -> P(x))

1. Which of the following is correct for predicate “All men drink coffee”?
   1. ∀x men(x) → (x, coffee)
   2. drink (x, coffee)→∀x men(x)
   3. **∀x men(x) → drink (x, coffee)**
   4. ∀men(x) → drink (x, coffee)
2. Which of the following is correct for predicate “Some boys play football”?
   1. **∃x boys(x) → play(x, football)**
   2. ∃ boys(x) → play(x, football)
   3. ∀x boys(x) → play(x, football)
   4. ∃x ∀boys(x) → play(x, football)
3. Dependent Type is used to encode \_\_\_\_\_\_\_\_\_\_\_\_ like "for all" and "there exists".
4. **Logic Quantifiers**
5. Analysing Quantifiers
6. Dependent Quantifiers
7. None of the above
8. “There exists x in the universe of discourse such that P(X) is true” is which Quantifiers statement?
9. Logical
10. Universal
11. **Existential**
12. None of these
13. What is the way to determine if a given function is dependant type

a. Result is independent of the argument

b. Result depends upon the usage in the program

**c. Result depends on the Value of its argument**

d. Result depends on available resources

1. Notation for an Existential Quantifier:

a.∀xP(x)

b.ΣP(x)

c.∅xP(x)

**d.∃xP(x)**

1. Representation of the following statement:

Every Clock has quartz

a. ∅xclock(x)->quartz(x)

b. ∃xclock(x)->quartz(x)

**c. ∀xclock(x)->quartz(x)**

d. none of the above

1. Representation of the following statement:

Some leaves are Red

**a. ∃xleaves(x)->red(x)**

b. ∀xleaves(x)->red(x)

c. Σleaves(x)->red(x)

d. none of the above

13. A function has dependent type if the \_\_\_\_\_\_of a function's result depends on the \_\_\_\_\_\_\_ of its

Argument

a. value and type

b. type and value

c. type and type

d. value and value

14. Dependent type paradigm used to encode logic's quantifiers like \_\_\_\_\_and \_\_\_\_\_\_\_

a. for one, there may exists

b. for all, there always

c. for all, there exists

d. for specific, there exists

15. Choose the correct one with respect to typing and typing-extensions library in python dependent type programming.

a. typing is not builtin python module where all possible types are defined.

typing\_extensions is an official package for new types in the future releases of python

b. typing is a builtin python module where all possible types are defined.

typing\_extensions is an official package for new types in the future releases of python

c. typing is a builtin python module where all possible types are defined.

typing\_extensions is not an official package for new types in the future releases of python

d. typing is not a builtin python module where all possible types are defined.

typing\_extensions is not an official package for new types in the future releases of python

16. Predict the output of the below mentioned python code without dependent type syntax:

def make\_hamburger(meat, number\_of\_meats):

return ["bread"] + [meat] \* number\_of\_meats + ["bread"]

print(make\_hamburger("ground beef", 2))

a. *['bread', 'ground beef', 2, 'bread']*

b. *['bread', 'ground beef', 'ground beef', 'bread']*

c. *TypeError: cannot concatenate 'str' and 'int' objects*

d. *['bread', 'ground beef', '2bread']*

17. Predict the output of the below mentioned python code with dependent type syntax:

**from** **typing** **import** List

**def** greet\_all(names: List[str]) -> None:

**for** name **in** names:

**print**('Hello ' + name)

names = ["Alice", "Bob", "Charlie"]

ages = [10, 20, 30]

greet\_all(names)

greet\_all(ages)

a.greet\_all(names) *# Ok!*

greet\_all(ages) *# Ok!*

b. a.greet\_all(names) *# Ok!*

greet\_all(ages) *# Error due to incompatible types*

c. a.greet\_all(names) *# Error due to incompatible types*

greet\_all(ages) *# Ok!*

d. a.greet\_all(names) *# Error due to incompatible types*

greet\_all(ages) *# Error due to incompatible types*

1. **Let x be a variable which refers to Universe of Disclosure such as x1,x2....xn then ,how to represent this statement using quantifiers “ All Man working in Industry”**
2. **∀x man(x) → work (x, Industry)** b)∀x man(x) → work(industry).
3. ∀x → work (x, industry) d)∀x man(x) → Industry(work)
4. **How do you represent the statement “Every man respects his parent.”**
5. ∀x woman(x) → respects (x, parent) **b) ∀x man(x) → respects (x, parent)**

c) ∀x man → respects (x, parent) d) ∀x man(x) → respects ( parent).

1. **How do you represent the statement “Some boys play cricket “**

a) ∃x boys(x) → play(all) **b**) ∀x boys(x) → play(x, cricket)

**c) ∃x boys(x) → play(x, cricket)** d)∃x^ ∀**x** boys(x) → play(x, cricket)