

TypeScript Code

```
1 import React, { useState } from 'react';
2 import { FileDown } from 'lucide-react';
3
4 const PythonSolutions = () => {
5   const [generating, setGenerating] = useState(false);
6
7   const solutions = [
8     {
9       no: "632",
10      question: "Implement the following hierarchy. The Book function has name, n (number of authors), authors (list of authors), publisher, ISBN, and year as its data members and the derived class has course as its data member.",
11      code: `class Book:
12        def __init__(self, name, n, authors, publisher, ISBN, year):
13          self.name = name
14          self.n = n
15          self.authors = authors
16          self.publisher = publisher
17          self.ISBN = ISBN
18          self.year = year
19
20        def display(self):
21          print(f"Name: {self.name}")
22          print(f"Number of authors: {self.n}")
23          print(f"Authors: {', '.join(self.authors)}")
24          print(f"Publisher: {self.publisher}")
25          print(f"ISBN: {self.ISBN}")
26          print(f"Year: {self.year}")
27
28        class CourseBook(Book):
29          def __init__(self, name, n, authors, publisher, ISBN, year, course):
30            super().__init__(name, n, authors, publisher, ISBN, year)
31            self.course = course
32
33          def display(self):
34            super().display()
35            print(f"Course: {self.course}")
36
37      # Example Usage
38      book = Book("The Great Gatsby", 1, ["F. Scott Fitzgerald"], "Scribner", "97807432735
65", 1925)
39      book.display()
40      print()
41      course_book = CourseBook("Data Science from Scratch", 1, ["Joel Grus"], "O'Reilly",
"9781492041139", 2019, "Data Science")
42      course_book.display()
43      },
44      {
45        no: "633",
46        question: "Implement the following hierarchy. The Staff function has name and salary as its data members, the derived class Teaching has subject as its data member and the class NonTeaching has department as its data member."
47      }
48    ],
49    {
50      no: "634",
51      question: "Implement the following hierarchy. The Staff function has name and salary as its data members, the derived class Teaching has subject as its data member and the class NonTeaching has department as its data member."
52    }
53  ],
54  {
55    no: "635",
56    question: "Implement the following hierarchy. The Staff function has name and salary as its data members, the derived class Teaching has subject as its data member and the class NonTeaching has department as its data member."
57  }
58]
```

```

47     code: `class Staff:
48         def __init__(self, name, salary):
49             self.name = name
50             self.salary = salary
51
52         def display(self):
53             print(f"Name: {self.name}")
54             print(f"Salary: {self.salary}")
55
56     class Teaching(Staff):
57         def __init__(self, name, salary, subject):
58             super().__init__(name, salary)
59             self.subject = subject
60
61         def display(self):
62             super().display()
63             print(f"Subject: {self.subject}")
64
65     class NonTeaching(Staff):
66         def __init__(self, name, salary, department):
67             super().__init__(name, salary)
68             self.department = department
69
70         def display(self):
71             super().display()
72             print(f"Department: {self.department}")
73
74     # Example Usage
75     staff_member = Staff("John", 50000)
76     teaching_member = Teaching("Jane", 60000, "Math")
77     non_teaching_member = NonTeaching("Joe", 40000, "Finance")
78
79     staff_member.display()
80     print()
81     teaching_member.display()
82     print()
83     non_teaching_member.display()
84     },
85     {
86         no: "634",
87         question: "Create a class called Student, having name and email as its data members and __init__(self, name, email) and putdata(self) as bound methods. Create another class called PhDguide having name, email, and students as its data members.",
88         code: `class Person:
89             def __init__(self, name, email):
90                 self.name = name
91                 self.email = email
92
93             def putdata(self):
94                 print("Name:", self.name)
95                 print("Email:", self.email)
96
97         class Student(Person):
98             def __init__(self, name, email):
99                 super().__init__(name, email)

```

```

100
101 class PhDguide(Person):
102     def __init__(self, name, email):
103         super().__init__(name, email)
104         self.students = []
105
106     def putdata(self):
107         super().putdata()
108         print("Students:", [s.name for s in self.students])
109
110     def add(self, student):
111         self.students.append(student)
112
113     def remove(self, student):
114         if student in self.students:
115             self.students.remove(student)
116             print(f"{student.name} has been removed from the list of students.")
117         else:
118             print(f"{student.name} is not in the list of students.")
119
120 # Example Usage
121 student1 = Student("John Doe", "johndoe@example.com")
122 student1.putdata()
123 print()
124
125 guide1 = PhDguide("Jane Smith", "janessmith@example.com")
126 guide1.putdata()
127 print()
128
129 guide1.add(student1)
130 guide1.putdata()
131 print()
132
133 guide1.remove(student1)
134 guide1.putdata()
135     },
136     {
137         no: "635",
138         question: "Program to demonstrate the issue of invoking __init__() in case of
multiple inheritance",
139         code: `class A:
140             def __init__(self):
141                 print("Class A __init__ called")
142                 self.a = "A"
143
144             class B:
145                 def __init__(self):
146                     print("Class B __init__ called")
147                     self.b = "B"
148
149             class C(A, B):
150                 def __init__(self):
151                     print("Class C __init__ called")
152                     A.__init__(self)
153                     B.__init__(self)

```

```

154         self.c = "C"
155
156     # Example Usage
157     obj = C()
158     print("\nObject attributes:")
159     print(f"a = {obj.a}")
160     print(f"b = {obj.b}")
161     print(f"c = {obj.c}```")
162     },
163     {
164         no: "636",
165         question: "Write program that has a class point. Define another class location which has two objects (Location and Destination) of class point. Also define function in Location that prints reflection of Destination on the x axis.",
166         code: `class Point:
167             def __init__(self, x, y):
168                 self.x = x
169                 self.y = y
170
171             class Destination(Point):
172                 pass
173
174             class Location(Point):
175                 def __init__(self, x1, y1, x2, y2):
176                     super().__init__(x1, y1)
177                     self.destination = Destination(x2, y2)
178
179                 def reflect_destination_on_x_axis(self):
180                     self.destination.y = -self.destination.y
181                     print(f"Reflected Destination point on X axis: ({self.destination.x}, {self.destination.y})")
182
183             # Example Usage
184             l = Location(1, 2, 3, 4)
185             l.reflect_destination_on_x_axis(```")
186             },
187             {
188                 no: "637",
189                 question: "Write a program that overload the + operator so that it can add two object of class fraction",
190                 code: `class Fraction:
191                     def __init__(self, numerator, denominator):
192                         self.numerator = numerator
193                         self.denominator = denominator
194
195                     def __add__(self, other):
196                         new_numerator = (self.numerator * other.denominator) + (other.numerator * se
197                         lf.denominator)
198                         new_denominator = self.denominator * other.denominator
199                         return Fraction(new_numerator, new_denominator)
200
201                     def display(self):
202                         print(f"{self.numerator}/{self.denominator}")
203
204             # Example Usage

```

```

204     f1 = Fraction(1, 2)
205     f2 = Fraction(1, 3)
206     f3 = f1 + f2
207
208     print("Fraction 1:", end=" ")
209     f1.display()
210     print("Fraction 2:", end=" ")
211     f2.display()
212     print("Sum:", end=" ")
213     f3.display()
214     },
215     {
216         no: "638",
217         question: "Write a program that overload the * operator so that it can add two
object of class fraction",
218         code: `class Fraction:
219             def __init__(self, numerator, denominator):
220                 self.numerator = numerator
221                 self.denominator = denominator
222
223             def __mul__(self, other):
224                 new_numerator = self.numerator * other.numerator
225                 new_denominator = self.denominator * other.denominator
226                 return Fraction(new_numerator, new_denominator)
227
228             def display(self):
229                 print(f"{self.numerator}/{self.denominator}")
230
231 # Example Usage
232 f1 = Fraction(2, 3)
233 f2 = Fraction(3, 4)
234 f3 = f1 * f2
235
236     print("Fraction 1:", end=" ")
237     f1.display()
238     print("Fraction 2:", end=" ")
239     f2.display()
240     print("Product:", end=" ")
241     f3.display()
242     },
243     {
244         no: "639",
245         question: "Write a program to find the distance between two points in cartesian
coordinate system",
246         code: `import math
247
248         class Point:
249             def __init__(self, x, y):
250                 self.x = x
251                 self.y = y
252
253             def distance(self, other):
254                 dx = self.x - other.x
255                 dy = self.y - other.y
256                 return math.sqrt(dx * dx + dy * dy)

```

```

257
258     # Example Usage
259     p1 = Point(3, 4)
260     p2 = Point(0, 0)
261     dist = p1.distance(p2)
262     print(f"Distance between points: {dist}```")
263     },
264     {
265         no: "640",
266         question: "Write a program to find the slope between two points in cartesian coordinate system",
267         code: `class Point:
268             def __init__(self, x, y):
269                 self.x = x
270                 self.y = y
271
272             def slope(self, other):
273                 if self.x == other.x:
274                     return "Undefined (vertical line)"
275                 return (other.y - self.y) / (other.x - self.x)
276
277     # Example Usage
278     p1 = Point(1, 2)
279     p2 = Point(4, 8)
280     slope_value = p1.slope(p2)
281     print(f"Slope between points: {slope_value}```"
282     },
283     {
284         no: "641",
285         question: "Create a class student with following member attributes: roll no, name, age and total marks. Write a python program to overload '==' operator to print the details of students having same marks.",
286         code: `class Student:
287             def __init__(self, roll_no, name, age, total_marks):
288                 self.roll_no = roll_no
289                 self.name = name
290                 self.age = age
291                 self.total_marks = total_marks
292
293             def display_student_info(self):
294                 print(f"Roll No: {self.roll_no}")
295                 print(f"Name: {self.name}")
296                 print(f"Age: {self.age}")
297                 print(f"Total Marks: {self.total_marks}")
298
299             def __eq__(self, other):
300                 if isinstance(other, Student):
301                     return self.total_marks == other.total_marks
302                 return False
303
304     # Example Usage
305     s1 = Student(1, "John", 20, 90)
306     s2 = Student(2, "Mary", 21, 90)
307     s3 = Student(3, "Peter", 19, 85)
308

```

```

309     if s1 == s2:
310         print(f"{s1.name} and {s2.name} have the same marks!")
311         s1.display_student_info()
312         print()
313         s2.display_student_info()
314     else:
315         print(f"{s1.name} and {s2.name} do not have the same marks.")
316     },
317     {
318         no: "642",
319         question: "Write a program to create a class called Data having 'value' as its
320         data member. Overload the (>) and the (<) operator for the class.",
321         code: `class Data:
322             def __init__(self, value):
323                 self.value = value
324
325             def __lt__(self, other):
326                 return self.value < other.value
327
328             def __gt__(self, other):
329                 return self.value > other.value
330
331             # Example Usage
332             d1 = Data(10)
333             d2 = Data(20)
334             d3 = Data(15)
335
336             print(f"d1 > d2: {d1 > d2}")
337             print(f"d2 > d3: {d2 > d3}")
338             print(f"d3 < d1: {d3 < d1}")
339             print(f"d3 < d2: {d3 < d2}")
340         },
341         {
342             no: "643",
343             question: "The following illustration creates a class called data. If no argum
344             ent is passed while instantiating the class a false is returned, otherwise a true is returne
345             d.",
346             code: `class Data:
347                 def __init__(self, value=None):
348                     if value:
349                         self.value = value
350                         self.status = True
351                     else:
352                         self.status = False
353
354                 def __str__(self):
355                     return f"status: {self.status}"
356
357             # Example Usage
358             data_obj1 = Data()
359             print(data_obj1)
360             print(data_obj2)`
361         },

```

```

361      {
362          no: "644",
363          question: "Create a 5X2 integer array from a range between 100 to 200 such tha
t the difference between each element is 10",
364          code: `import numpy as np
365
366      # Create array with values from 100 to 200 with step 10
367      arr = np.arange(100, 200, 10)
368
369      # Reshape to 5x2
370      result = arr.reshape(5, 2)
371
372      print("5x2 Array:")
373      print(result)`
374      },
375      {
376          no: "645",
377          question: "Following is the provided numpy array. Return array of items by tak
ing the third column from all rows",
378          code: `import numpy as np
379
380      # Sample array
381      sampleArray = np.array([[11, 22, 33], [44, 55, 66], [77, 88, 99]])
382
383      print("Original Array:")
384      print(sampleArray)
385
386      # Get third column from all rows
387      third_column = sampleArray[:, 2]
388
389      print("\nThird column:")
390      print(third_column)`
391      },
392      {
393          no: "646",
394          question: "Return array of odd rows and even columns from below numpy array",
395          code: `import numpy as np
396
397      # Sample array
398      sampleArray = np.array([[3, 6, 9, 12], [15, 18, 21, 24], [27, 30, 33, 36], [39, 42,
45, 48], [51, 54, 57, 60]])
399
400      print("Original Array:")
401      print(sampleArray)
402
403      # Get odd rows (index 0, 2, 4) and even columns (index 0, 2)
404      result = sampleArray[::2, ::2]
405
406      print("\nOdd rows and even columns:")
407      print(result)`
408      },
409      {
410          no: "647",
411          question: "Sort following NumPy array - Case 1: Sort array by the second row,
Case 2: Sort the array by the second column",

```

```

412         code: `import numpy as np
413
414     # Sample array
415     sampleArray = np.array([[34, 43, 73], [82, 22, 12], [53, 94, 66]])
416
417     print("Original Array:")
418     print(sampleArray)
419
420     # Case 1: Sort by second row
421     sorted_by_row = sampleArray[:, sampleArray[1, :].argsort()]
422     print("\nCase 1 - Sorted by second row:")
423     print(sorted_by_row)
424
425     # Case 2: Sort by second column
426     sorted_by_column = sampleArray[sampleArray[:, 1].argsort()]
427     print("\nCase 2 - Sorted by second column:")
428     print(sorted_by_column)
429
430     },
431     {
432         no: "648",
433         question: "Print max from axis 0 and min from axis 1 from the following 2-D ar
ray",
434         code: `import numpy as np
435
436     # Sample array
437     sampleArray = np.array([[34, 43, 73], [82, 22, 12], [53, 94, 66]])
438
439     print("Original Array:")
440     print(sampleArray)
441
442     # Max from axis 0 (column-wise max)
443     max_axis_0 = np.max(sampleArray, axis=0)
444     print("\nMax from axis 0:")
445     print(max_axis_0)
446
447     # Min from axis 1 (row-wise min)
448     min_axis_1 = np.min(sampleArray, axis=1)
449     print("\nMin from axis 1:")
450     print(min_axis_1)
451
452     },
453     {
454         no: "649",
455         question: "Write a NumPy array program to convert the values of Fahrenheit deg
rees into Celsius degrees",
456         code: `import numpy as np
457
458     # Fahrenheit values
459     fahrenheit = np.array([0, 12, 45.21, 34, 99.91, 32])
460
461     print("Values in Fahrenheit degrees:")
462     print(fahrenheit)
463
464     # Convert to Celsius using formula: C = (F - 32) * 5/9
465     celsius = (fahrenheit - 32) * 5 / 9
466

```

```

465     print("\nValues in Centigrade degrees:")
466     print(celsius)
467
468     # Sort the array
469     celsius_sorted = np.sort(celsius)
470     print("\nSorted Celsius values:")
471     print(celsius_sorted)
472     },
473     {
474         no: "650",
475         question: "Import numpy as np. arr = np.array([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
476 11, 12]]). Reshape arr into a 2D array and a 3D array.",
477         code: `import numpy as np
478
479     # Original 1D array
480     arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
481
482     print("Original 1D array:")
483     print(arr)
484
485     # Reshape to 2D array (3x4)
486     arr_2d = arr.reshape(3, 4)
487     print("\n2D Array (3x4):")
488     print(arr_2d)
489
490     # Reshape to 3D array (2x2x3)
491     arr_3d = arr.reshape(2, 2, 3)
492     print("\n3D Array (2x2x3):")
493     print(arr_3d)
494     },
495     {
496         no: "651",
497         question: "Imagine you own a call center. Use the following abstract class tem-
plate to create three more classes: Respondent, Manager, and Director that inherit the Empl-
oyee Abstract Class.",
498         code: `from abc import ABC, abstractmethod
499
500     class EmployeeABC(ABC):
501         @abstractmethod
502         def receive_call(self):
503             pass
504
505         @abstractmethod
506         def end_call(self):
507             pass
508
509         @abstractmethod
510         def is_free(self):
511             pass
512
513         @abstractmethod
514         def get_rank(self):
515             pass
516
517     class Respondent(EmployeeABC):

```

```
517     def __init__(self, name):
518         self.name = name
519         self.free = True
520
521     def receive_call(self):
522         print(f"{self.name} (Respondent) receiving call")
523         self.free = False
524
525     def end_call(self):
526         print(f"{self.name} (Respondent) ending call")
527         self.free = True
528
529     def is_free(self):
530         return self.free
531
532     def get_rank(self):
533         return "Respondent"
534
535 class Manager(EmployeeABC):
536     def __init__(self, name):
537         self.name = name
538         self.free = True
539
540     def receive_call(self):
541         print(f"{self.name} (Manager) receiving call")
542         self.free = False
543
544     def end_call(self):
545         print(f"{self.name} (Manager) ending call")
546         self.free = True
547
548     def is_free(self):
549         return self.free
550
551     def get_rank(self):
552         return "Manager"
553
554 class Director(EmployeeABC):
555     def __init__(self, name):
556         self.name = name
557         self.free = True
558
559     def receive_call(self):
560         print(f"{self.name} (Director) receiving call")
561         self.free = False
562
563     def end_call(self):
564         print(f"{self.name} (Director) ending call")
565         self.free = True
566
567     def is_free(self):
568         return self.free
569
570     def get_rank(self):
571         return "Director"
```

```

572
573     # Example Usage
574     r1 = Respondent("John")
575     m1 = Manager("Sarah")
576     d1 = Director("Michael")
577
578     r1.receive_call()
579     print(f"Is {r1.name} free? {r1.is_free()}")
580     r1.end_call()
581     print(f"Is {r1.name} free? {r1.is_free()}``)
582         },
583     {
584         no: "652",
585         question: "Write a python program to create a Bus child class that inherits from the Vehicle class. The default fare charge of any vehicle is seating capacity * 100. If Vehicle is Bus instance, we need to add an extra 10% on full fare as a maintenance charge.",
586         code: `class Vehicle:
587             def __init__(self, seating_capacity):
588                 self.seating_capacity = seating_capacity
589
590             def fare(self):
591                 return self.seating_capacity * 100
592
593         class Bus(Vehicle):
594             def fare(self):
595                 base_fare = super().fare()
596                 maintenance_charge = base_fare * 0.10
597                 total_fare = base_fare + maintenance_charge
598                 return total_fare
599
600         # Example Usage
601         v1 = Vehicle(50)
602         print(f"Vehicle fare: {v1.fare()}")
603
604         b1 = Bus(50)
605         print(f"Bus fare (with 10% maintenance): {b1.fare()}")
606
607         # Example with car
608         c1 = Vehicle(5)
609         print(f"\nCar fare: {c1.fare()}```)
610     },
611     {
612         no: "653",
613         question: "Create an abstract class named Shape. Create an abstract method named calculate_area for the Shape class. Create two classes Rectangle and Circle that inherit from Shape class. Implement calculate_area method in both classes.",
614         code: `from abc import ABC, abstractmethod
615         import math
616
617         class Shape(ABC):
618             @abstractmethod
619             def calculate_area(self):
620                 pass
621
622         class Rectangle(Shape):

```

```

623     def __init__(self, length, breadth):
624         self.length = length
625         self.breadth = breadth
626
627     def calculate_area(self):
628         return self.length * self.breadth
629
630 class Circle(Shape):
631     def __init__(self, radius):
632         self.radius = radius
633
634     def calculate_area(self):
635         return math.pi * self.radius * self.radius
636
637 # Example Usage
638 rect = Rectangle(10, 5)
639 print(f"Area of Rectangle: {rect.calculate_area()}")
640
641 circle = Circle(7)
642 print(f"Area of Circle: {circle.calculate_area():.2f}")
643
644 # Check if Rectangle area is greater than Circle area
645 if rect.calculate_area() > circle.calculate_area():
646     print("\nRectangle has greater area")
647 else:
648     print("\nCircle has greater area")
649 },
650 {
651     no: "654",
652     question: "Write a python program to demonstrate the use of super() method to
call the method of base class.",
653     code: `class Parent:
654         def show(self):
655             print("Inside Parent class method")
656
657     class Child(Parent):
658         def show(self):
659             print("Inside Child class method")
660             super().show()
661             print("Back to Child class method")
662
663 # Example Usage
664 obj = Child()
665 obj.show()
666 },
667 {
668     no: "655",
669     question: "Create a class called Matrix containing constructor that initialize
d the number of rows and number of columns of a new Matrix object.",
670     code: `class Matrix:
671         def __init__(self, rows, cols):
672             self.rows = rows
673             self.cols = cols
674             self.data = [[0 for j in range(cols)] for i in range(rows)]
675

```

```

676     def set_value(self, row, col, value):
677         if 0 <= row < self.rows and 0 <= col < self.cols:
678             self.data[row][col] = value
679         else:
680             print("Invalid position")
681
682     def get_value(self, row, col):
683         if 0 <= row < self.rows and 0 <= col < self.cols:
684             return self.data[row][col]
685         return None
686
687     def display(self):
688         for row in self.data:
689             print(row)
690
691 # Example Usage
692 m = Matrix(3, 3)
693 m.set_value(0, 0, 1)
694 m.set_value(0, 1, 2)
695 m.set_value(0, 2, 3)
696 m.set_value(1, 0, 4)
697 m.set_value(1, 1, 5)
698 m.set_value(1, 2, 6)
699 m.set_value(2, 0, 7)
700 m.set_value(2, 1, 8)
701 m.set_value(2, 2, 9)
702
703 print("Matrix:")
704 m.display()
705     },
706     {
707         no: "656",
708         question: "Find the MRO of class Z of below program",
709         code: `class A:
710             pass
711
712             class B:
713                 pass
714
715             class C:
716                 pass
717
718             class D:
719                 pass
720
721             class E:
722                 pass
723
724             class K1(C, A, B):
725                 pass
726
727             class K3(A, D):
728                 pass
729
730             class K2(B, D, E):

```

```

731     pass
732
733 class Z(K1, K3, K2):
734     pass
735
736 # Find MRO (Method Resolution Order)
737 print("MRO of class Z:")
738 print(Z.__mro__)
739
740 print("\nMRO in readable format:")
741 for i, cls in enumerate(Z.__mro__):
742     print(f"{i+1}. {cls.__name__}`"
743     ),
744 {
745     no: "657",
746     question: "Write a Python Program to find the Net Salary of Employee using Inheritance. Create three class Employee, Perks, NetSalary. Make an Employee class as an abstract class.",
747     code: `from abc import ABC, abstractmethod
748
749 class Employee(ABC):
750     def __init__(self, emp_id, name, salary):
751         self.emp_id = emp_id
752         self.name = name
753         self.salary = salary
754
755     @abstractmethod
756     def print_details(self):
757         pass
758
759     @abstractmethod
760     def get_salary(self):
761         pass
762
763 class Perks:
764     def __init__(self, da, hra, pf):
765         self.da = da
766         self.hra = hra
767         self.pf = pf
768
769     def calculate_total_perks(self, basic_salary):
770         da_amount = basic_salary * (self.da / 100)
771         hra_amount = basic_salary * (self.hra / 100)
772         pf_amount = basic_salary * (self.pf / 100)
773         return da_amount + hra_amount - pf_amount
774
775 class NetSalary(Employee):
776     def __init__(self, emp_id, name, salary, da, hra, pf):
777         Employee.__init__(self, emp_id, name, salary)
778         self.perks = Perks(da, hra, pf)
779
780     def print_details(self):
781         print(f"Employee ID: {self.emp_id}")
782         print(f"Employee Name: {self.name}")
783         print(f"Basic Salary: {self.salary}")

```

```

784
785     def get_salary(self):
786         perks_amount = self.perks.calculate_total_perks(self.salary)
787         net_salary = self.salary + perks_amount
788         return net_salary
789
790     # Example Usage
791     emp = NetSalary(1, "John", 25000, 35, 17, 12)
792     emp.print_details()
793     print(f"DA: 35%")
794     print(f"HRA: 17%")
795     print(f"PF: 12%")
796     print(f"Total Salary: {emp.get_salary():.2f}```"
797         )
798     ];
799
800     const generatePDF = () => {
801       setGenerating(true);
802
803       setTimeout(() => {
804         const content = document.getElementById('solutions-content');
805         const printWindow = window.open('', '', 'height=600,width=800');
806
807         printWindow.document.write('<html><head><title>Python Programming Solutions - Unit 9</title>');
808         printWindow.document.write('<style>');
809         printWindow.document.write(`

body { font-family: Arial, sans-serif; padding: 20px; line-height: 1.6; }
h1 { color: #2563eb; text-align: center; border-bottom: 3px solid #2563eb; padding-bottom: 10px; }
h2 { color: #1e40af; margin-top: 30px; background: #eff6ff; padding: 10px; border-left: 4px solid #2563eb; }
.question { background: #f3f4f6; padding: 10px; margin: 10px 0; border-radius: 5px; }
pre { background: #1e293b; color: #e2e8f0; padding: 15px; border-radius: 5px; overflow-x: auto; font-size: 12px; }
code { font-family: 'Courier New', monospace; }
.page-break { page-break-after: always; }

`);
810         printWindow.document.write('</style></head><body>');
811         printWindow.document.write(`<h1>Python Programming Solutions - Unit 9</h1>`);
812         printWindow.document.write(`<p style="text-align: center; color: #666;">Complete Solutions with Question Numbers</p>`);
813
814         solutions.forEach((sol, idx) => {
815           printWindow.document.write(`<h2>Question ${sol.no}</h2>`);
816           printWindow.document.write(`<div class="question"><strong>Problem:</strong> ${sol.question}</div>`);
817           printWindow.document.write(`<pre><code>${sol.code.replace(/</g, '&lt;').replace(>/g, '&gt;')}</code></pre>`);
818           if ((idx + 1) % 3 === 0 && idx < solutions.length - 1) {
819             printWindow.document.write(`<div class="page-break"></div>`);
820           }
821         });
822       });
823     };

```

```

831     printWindow.document.write('</body></html>');
832     printWindow.document.close();
833
834     printWindow.onload = function() {
835         printWindow.focus();
836         printWindow.print();
837         setGenerating(false);
838     };
839 }, 500);
840 };
841
842 return (
843     <div className="min-h-screen bg-gradient-to-br from-blue-50 to-indigo-100 p-8">
844         <div className="max-w-6xl mx-auto">
845             <div className="bg-white rounded-lg shadow-xl p-8 mb-6">
846                 <h1 className="text-4xl font-bold text-blue-600 mb-2 text-center">
847                     Python Programming Solutions
848                 </h1>
849                 <p className="text-gray-600 text-center mb-6">Unit 9 - Complete Solutions
</p>
850
851             <div className="flex justify-center mb-8">
852                 <button
853                     onClick={generatePDF}
854                     disabled={generating}
855                     className="bg-blue-600 hover:bg-blue-700 text-white font-bold py-3 px-6 rounded-lg flex items-center gap-2 transition-all disabled:opacity-50 disabled:cursor-not-allowed">
856                     >
857                         <FileDown size={20} />
858                         {generating ? 'Generating PDF...' : 'Download as PDF'}
859                     </button>
860                 </div>
861             </div>
862
863             <div id="solutions-content" className="bg-white rounded-lg shadow-xl p-8">
864                 {solutions.map((sol, idx) => (
865                     <div key={idx} className="mb-12 pb-8 border-b border-gray-200 last:border-b-0">
866                         <h2 className="text-2xl font-bold text-blue-700 mb-3">
867                             Question {sol.no}
868                         </h2>
869                         <div className="bg-gray-50 p-4 rounded-lg mb-4">
870                             <p className="text-gray-700">{sol.question}</p>
871                         </div>
872                         <div className="bg-gray-900 rounded-lg overflow-hidden">
873                             <div className="bg-gray-800 px-4 py-2 text-sm text-gray-300">
874                                 Python Code
875                             </div>
876                             <pre className="p-4 overflow-x-auto">
877                                 <code className="text-sm text-gray-100">{sol.code}</code>
878                             </pre>
879                         </div>
880                     </div>
881                 )));

```

```
882         </div>
883
884     <div className="bg-white rounded-lg shadow-xl p-6 mt-6 text-center">
885         <p className="text-gray-600">
886              Total Solutions: <span className="font-bold text-blue-600">{solution
s.length}</span>
887         </p>
888         <p className="text-sm text-gray-500 mt-2">
889             All solutions use basic Python syntax for easy understanding
890         </p>
891         </div>
892     </div>
893     </div>
894     );
895 };
896
897 export default PythonSolutions;
```