Assignment-7.1

R.Nikhil Kumar

2303A52260

Batch – 44

Task Description #1 (Syntax Errors – Missing Parentheses in Print Statement)

Task: Provide a Python snippet with a missing parenthesis in a print

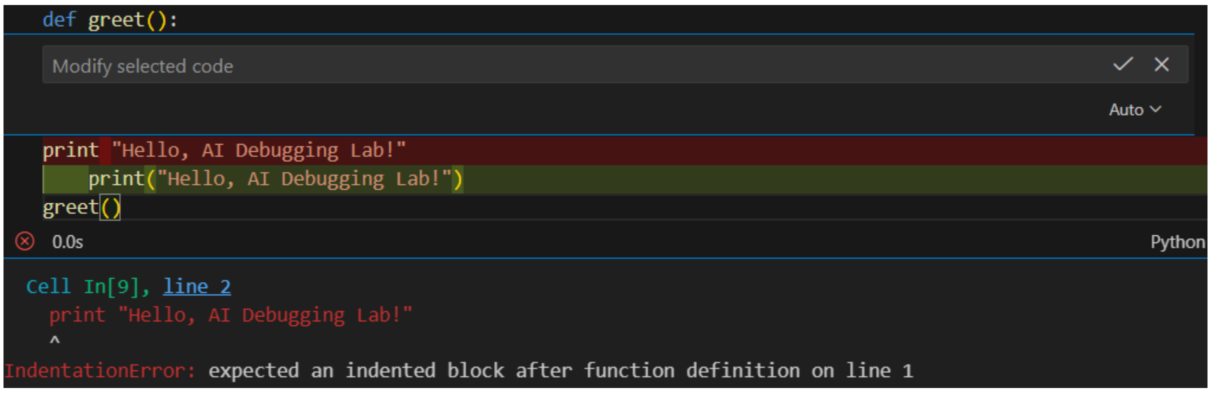
statement (e.g., print "Hello"). Use AI to detect and fix the syntax error.

# Bug: Missing parentheses in print statement

def greet():

print "Hello, AI Debugging Lab!"

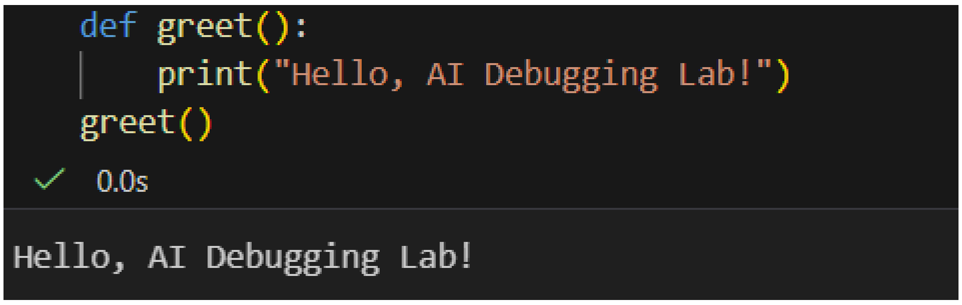
greet()



```
def greet():
Modify selected code
Auto
print "Hello, AI Debugging Lab!"
    print("Hello, AI Debugging Lab!")
greet()
0.0s                                                    Python

Cell In[9], line 2
  print "Hello, AI Debugging Lab!"
  ^
IndentationError: expected an indented block after function definition on line 1
```

• ***Corrected code with proper syntax and AI explanation.***



```
def greet():
    print("Hello, AI Debugging Lab!")
greet()
0.0s

Hello, AI Debugging Lab!
```

Task Description #2 (Incorrect condition in an If Statement)

Task: Supply a function where an if-condition mistakenly uses =

instead of ==. Let AI identify and fix the issue.

# Bug: Using assignment (=) instead of comparison (==)

def check_number(n):

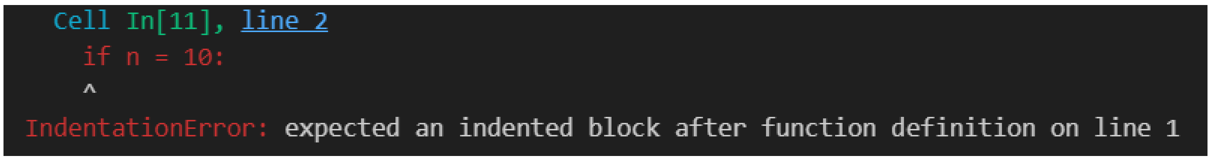
if n = 10:

return "Ten"

else:

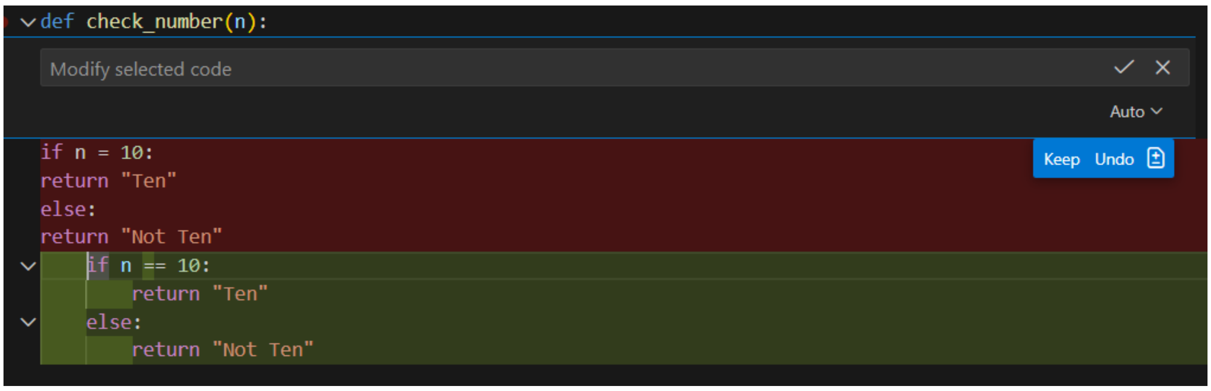
return "Not Ten"

#Output



```
  Cell In[11], line 2
    if n = 10:
    ^
IndentationError: expected an indented block after function definition on line 1
```

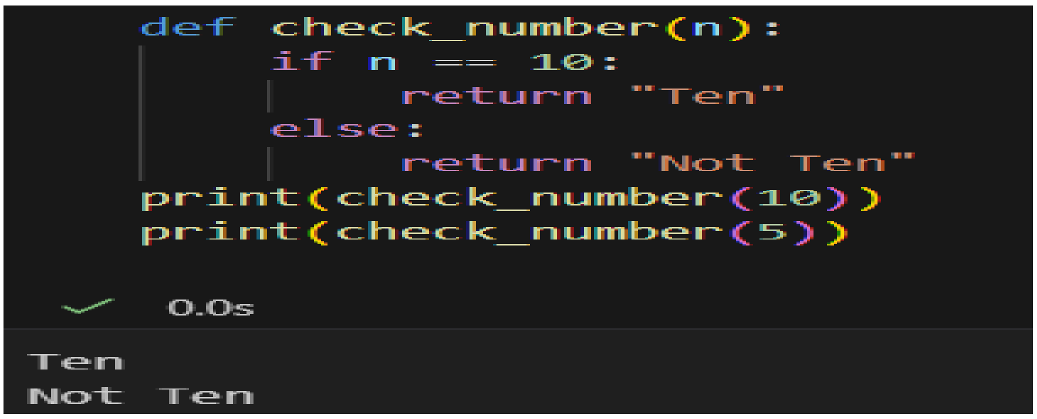
Corrected code using == with explanation and successful test execution



```
def check_number(n):
```

```
if n = 10:
return "Ten"
else:
return "Not Ten"
    if n == 10:
        return "Ten"
    else:
        return "Not Ten"
```

#output:



```
def check_number(n):
    if n == 10:
        return "Ten"
    else:
        return "Not Ten"
print(check_number(10))
print(check_number(5))
```

```
Ten
Not Ten
```

Task Description #3 (Runtime Error – File Not Found)

Task: Provide code that attempts to open a non-existent file and crashes. Use AI to apply safe error handling.

# Bug: Program crashes if file is missing

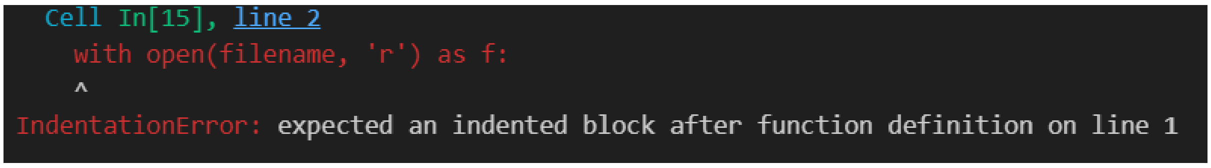
def read_file(filename):

with open(filename, 'r') as f:

return f.read()

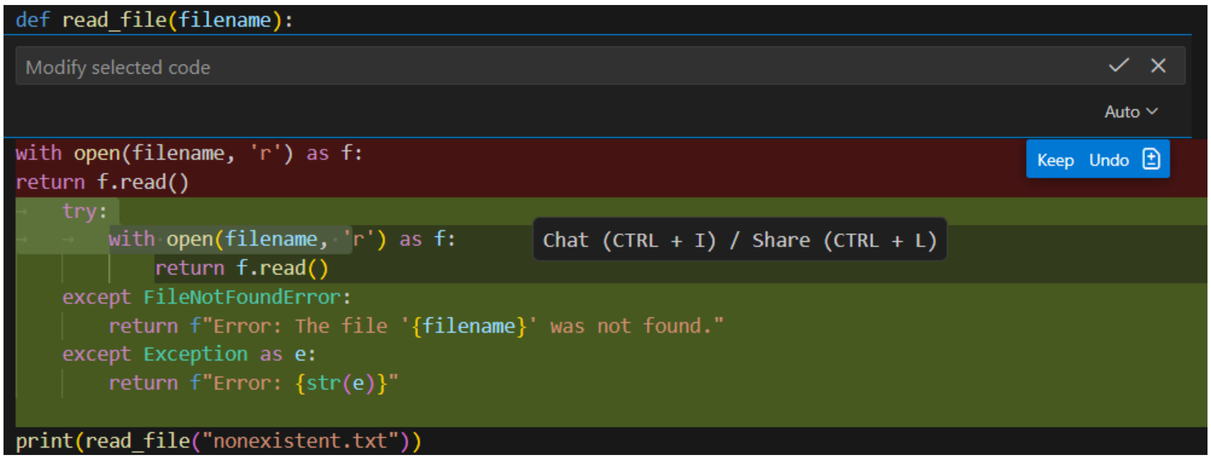
print(read_file("nonexistent.txt"))

#Output :



```
  Cell In[15], line 2
    with open(filename, 'r') as f:
    ^
IndentationError: expected an indented block after function definition on line 1
```

Safe file handling with exception management.

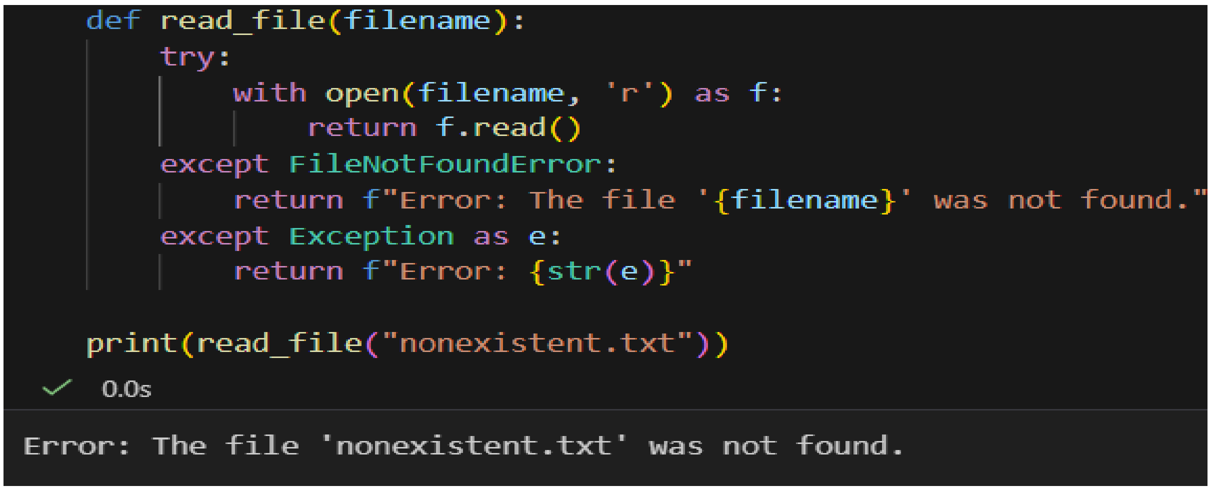


```
def read_file(filename):
```

```
with open(filename, 'r') as f:
return f.read()
    try:
        with open(filename, 'r') as f:
            return f.read()
    except FileNotFoundError:
        return f"Error: The file '{filename}' was not found."
    except Exception as e:
        return f"Error: {str(e)}"

print(read_file("nonexistent.txt"))
```

#Output :



```
def read_file(filename):
    try:
        with open(filename, 'r') as f:
            return f.read()
    except FileNotFoundError:
        return f"Error: The file '{filename}' was not found."
    except Exception as e:
        return f"Error: {str(e)}"

print(read_file("nonexistent.txt"))
```

✓ 0.0s

```
Error: The file 'nonexistent.txt' was not found.
```

Task Description #4 (Calling a Non-Existent Method)

Task: Give a class where a non-existent method is called (e.g., obj.undefined_method()). Use AI to debug and fix.

# Bug: Calling an undefined method

class Car:

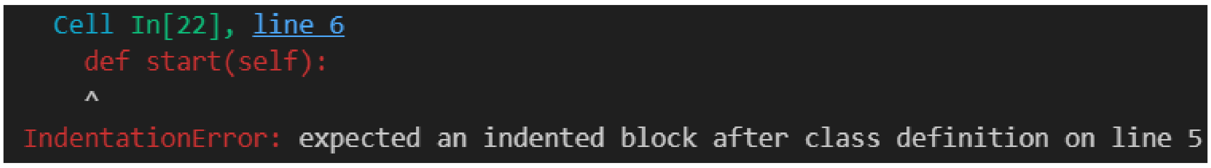
def start(self):

return "Car started"

my_car = Car()

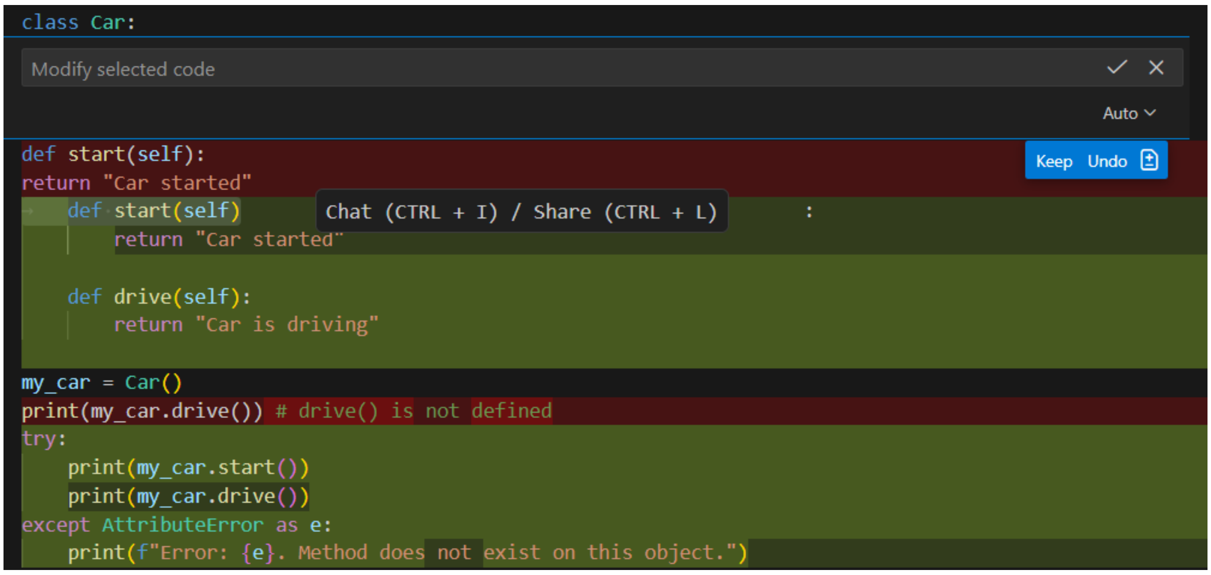
print(my_car.drive()) # drive() is not defined

#Output :



```
  Cell In[22], line 6
    def start(self):
    ^
IndentationError: expected an indented block after class definition on line 5
```

Corrected class with clear AI explanation

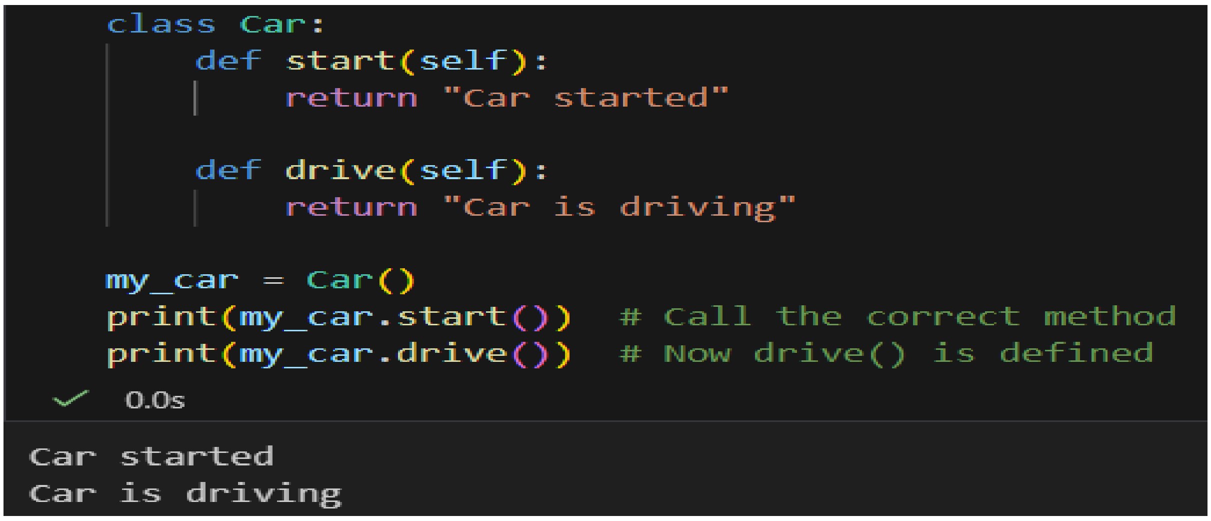


```
class Car:
def start(self):
return "Car started"
    def start(self)
        return "Car started"

    def drive(self):
        return "Car is driving"

my_car = Car()
print(my_car.drive()) # drive() is not defined
try:
    print(my_car.start())
    print(my_car.drive())
except AttributeError as e:
    print(f"Error: {e}. Method does not exist on this object.")
```

#Output :



```
class Car:
    def start(self):
        return "Car started"

    def drive(self):
        return "Car is driving"

my_car = Car()
print(my_car.start())  # Call the correct method
print(my_car.drive())  # Now drive() is defined
```

```
✓ 0.0s
```

```
Car started
Car is driving
```

Task Description #5 (TypeError – Mixing Strings and Integers in Addition)

Task: Provide code that adds an integer and string ("5" + 2) causing a ***TypeError****. Use AI to resolve the bug.*

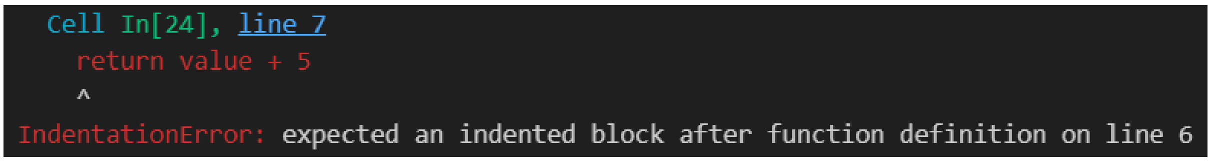
# Bug: TypeError due to mixing string and integer

def add_five(value):

return value + 5

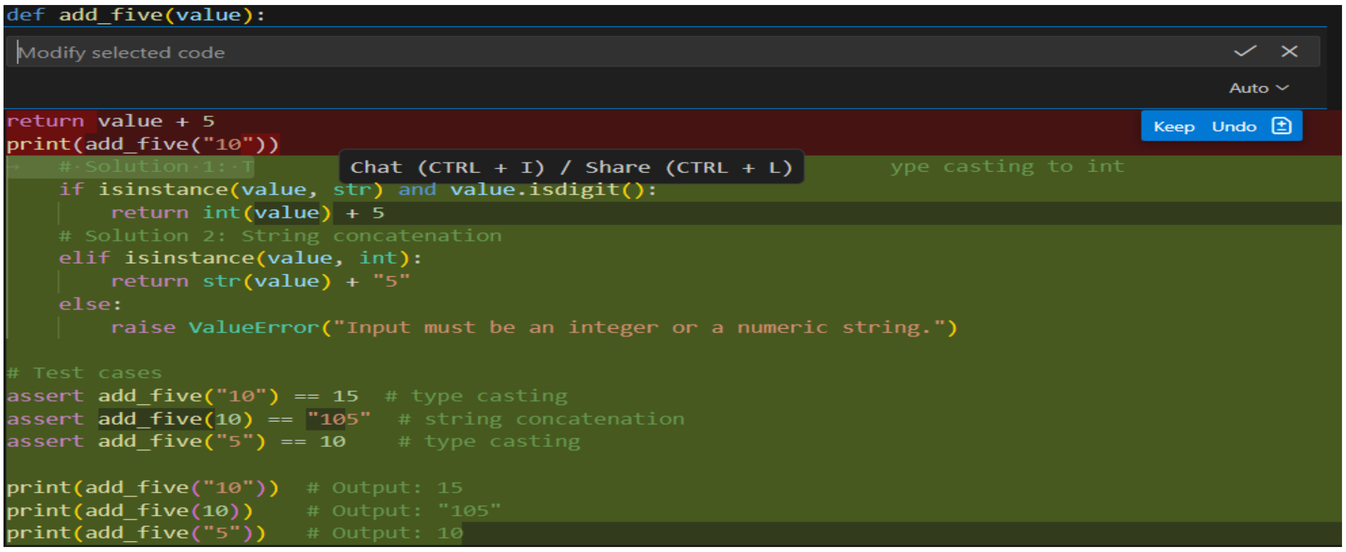
print(add_five("10"))

#Output :



```
  Cell In[24], line 7
    return value + 5
    ^
IndentationError: expected an indented block after function definition on line 6
```

Corrected code that runs successfully for multiple inputs.

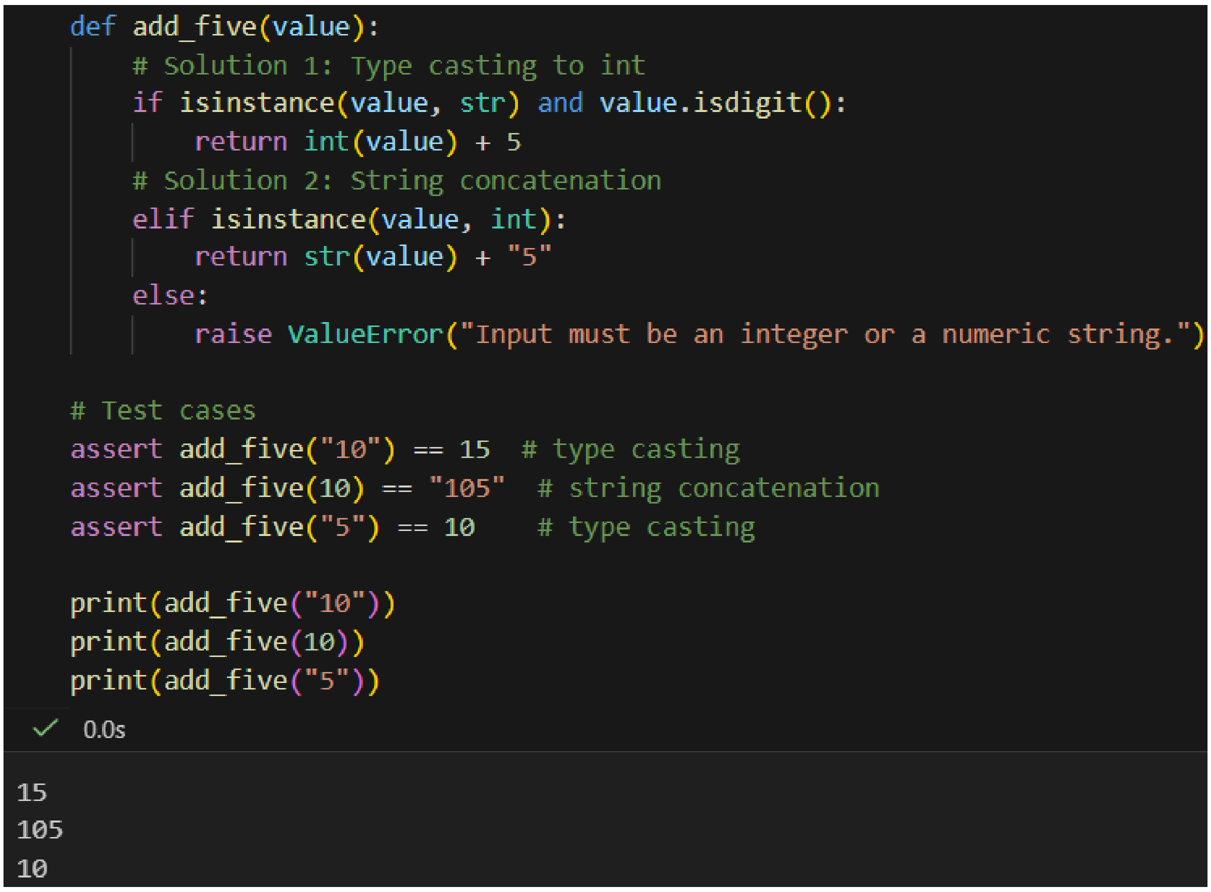


```
def add_five(value):
```

```
return value + 5
print(add_five("10"))
    # Solution 1: T                                     ype casting to int
    if isinstance(value, str) and value.isdigit():
        return int(value) + 5
    # Solution 2: String concatenation
    elif isinstance(value, int):
        return str(value) + "5"
    else:
        raise ValueError("Input must be an integer or a numeric string.")

# Test cases
assert add_five("10") == 15  # type casting
assert add_five(10) == "105"  # string concatenation
assert add_five("5") == 10    # type casting

print(add_five("10"))  # Output: 15
print(add_five(10))    # Output: "105"
print(add_five("5"))   # Output: 10
```



```
def add_five(value):
    # Solution 1: Type casting to int
    if isinstance(value, str) and value.isdigit():
        return int(value) + 5
    # Solution 2: String concatenation
    elif isinstance(value, int):
        return str(value) + "5"
    else:
        raise ValueError("Input must be an integer or a numeric string.")

# Test cases
assert add_five("10") == 15  # type casting
assert add_five(10) == "105"  # string concatenation
assert add_five("5") == 10    # type casting

print(add_five("10"))
print(add_five(10))
print(add_five("5"))
```

✓ 0.0s

```
15
105
10
```