

Problem 1

```
In [ ]: 1 %%typecheck
        2 import typing as tp
        3
        4 def foo(a: tp.Optional[int]) -> int:
        5     return a + 1
        6
        7 a: int = {"a": 1}.get("a")
        8 foo(a)
```

Problem 2

```
In [ ]: 1 %%typecheck
        2 import typing as tp
        3
        4 aa = [1, "2", [3]]
        5 aa[0] += 1
        6 aa.append({1, 3})
```

Problem 3

```
In [ ]: 1 %%typecheck
        2 import typing as tp
        3
        4
        5 def foo(a: tp.Set[int]) -> None:
        6     pass
        7
        8 a = {1.1, 2.1, 3.1}
        9 foo(a)
       10 b = {1, 2, 3}
       11 foo(b)
       12 c = {True, False}
       13 foo(c)
```

Problem 4

```
In [ ]: 1 %%typecheck
        2 import typing as tp
        3
        4
        5 def foo(a: tp.AbstractSet[int]) -> None:
        6     pass
        7
        8 a = {1.1, 2.1, 3.1}
        9 foo(a)
       10 b = {1, 2, 3}
       11 foo(b)
       12 c = {True, False}
       13 foo(c)
```

Problem 5

```
In [ ]: 1 %%typecheck
        2 import typing as tp
        3
        4 class A:
        5     pass
        6
        7 class B(A):
        8     pass
        9
       10 def foo(a: A) -> B:
       11     return a.__class__()
       12
       13
       14 foo(A())
       15 foo(B())
```

Problem 6

In []:

```
1 %%typecheck
2 class T:
3     pass
4
5 class S(T):
6     pass
7
8 class A:
9     VAR = T()
10
11 class B(A):
12     VAR = S()
13
14 class C(B):
15     VAR = T()
16
```

Problem 7

```
In [ ]: 1 %%typecheck
        2 import typing as tp
        3
        4 class A:
        5     pass
        6
        7 class B(A):
        8     pass
        9
        10 def g(f: tp.Callable[[A], B]) -> None:
        11     pass
        12
        13 def f1(a: A) -> A:
        14     pass
        15
        16 def f2(a: A) -> B:
        17     pass
        18
        19 def f3(a: B) -> A:
        20     pass
        21
        22 def f4(a: B) -> B:
        23     pass
        24
        25 g(f1)
        26 g(f2)
        27 g(f3)
        28 g(f4)
```

Problem 8

```
In [ ]: 1 %%typecheck
2 import typing as tp
3
4
5 def foo(a: tp.Iterable[str]) -> bool:
6     b = len(a)
7     c = sum(1 for i in a)
8     return b == c
9
10 foo(["a", "b"])
11 foo("ab")
12 foo({"a": 2})
13
14 class A:
15     def __len__(self) -> int:
16         return 1
17
18 foo(A())
19
20 class B:
21     def __iter__(self) -> tp.Iterator[int]:
22         return iter([])
23
24 foo(B())
25
26
27 class C:
28     def __iter__(self) -> tp.Iterator[str]:
29         return iter([])
30
31 foo(C())
```

Problem 9

```
In [ ]: 1 %%typecheck
        2 import typing as tp
        3
        4 class Fooable(tp.Protocol, tp.Sized):
        5     def foo(self, a: int) -> None:
        6         pass
        7
        8
        9 class A(Fooable):
        10     def __len__(self) -> int:
        11         return 10
        12
        13 class B:
        14     def foo(self, a: int) -> None:
        15         pass
        16
        17     def __len__(self) -> int:
        18         return 10
        19
        20 class C:
        21     def foo(self, a: int) -> None:
        22         pass
        23
        24
        25 def foo(a: int) -> None:
        26     pass
        27
        28
        29 def f(c: Fooable) -> None:
        30     c.foo(1)
        31     len(c)
        32     "10" in c
        33
        34 f([])
        35 f(A())
        36 f(B())
        37 f(C())
        38 f(foo)
```

Problem 10

```
In [ ]: 1 %%typecheck
2 import typing as tp
3
4 T = tp.TypeVar("T", bound=tp.SupportsFloat, covariant=True)
5
6 class A(tp.Generic[T]):
7     def __init__(self, a: T) -> None:
8         self._a: tp.SupportsFloat = a
9
10    def increment(self) -> float:
11        self._a = float(self._a) + 1
12        return self._a
13
14
15 A(1)
16 A(1.2)
17 A(True)
18 A("1.3")
19
20 class B:
21     def __float__(self) -> float:
22         return 1.1
23
24 A(B())
25
26 def g(a: A[int]) -> None:
27     pass
28
29 g(A(1.4))
30 g(A(True))
```