-   Python

Table of contents

1

```python
#
def greet(name):
    return f"Hello, {name}!"

#
def greet_typed(name: str) -> str:
    return f"Hello, {name}!"

#
message = greet_typed(" ")
print(message)
```

```
Hello,  !
```

2

2.1

```
from typing import List, Dict, Tuple, Optional

#
name: str = " "
age: int = 25
height: float = 170.5
is_student: bool = True

print(f" : {name} ({type(name).__name__})")
print(f" : {age} ({type(age).__name__})")
print(f" : {height} ({type(height).__name__})")
print(f" : {is_student} ({type(is_student).__name__})")
```

```
 :   (str)
 : 25 (int)
 : 170.5 (float)
 : True (bool)
```

2.2

```
#
numbers: List[int] = [1, 2, 3, 4, 5]
names: List[str] = [" ", " ", " "]

#
student_ages: Dict[str, int] = {
    " ": 20,
    " ": 22,
    " ": 19
}

#
coordinates: Tuple[float, float] = (35.6762, 139.6503)
rgb_color: Tuple[int, int, int] = (255, 128, 0)

print(f"   : {numbers}")
print(f"   : {names}")
print(f"   : {student_ages}")
print(f" : {coordinates}")
print(f"RGB : {rgb_color}")
```

```
   : [1, 2, 3, 4, 5]
   : [' ', ' ', ' ']
   : {' ': 20, ' ': 22, ' ': 19}
 : (35.6762, 139.6503)
RGB : (255, 128, 0)
```

3

3.1

```python
def add_numbers(a: int, b: int) -> int:
    """       """
    return a + b

def calculate_average(numbers: List[float]) -> float:
    """      """
    if not numbers:
        return 0.0
    return sum(numbers) / len(numbers)

def format_name(first: str, last: str, middle: str = "") -> str:
    """     """
    if middle:
        return f"{last} {first} {middle}"
    return f"{last} {first}"

#
result = add_numbers(10, 20)
print(f"10 + 20 = {result}")

scores = [85.5, 92.0, 78.5, 90.0]
avg = calculate_average(scores)
print(f" : {avg:.1f}")

full_name = format_name(" ", " ")
print(f"  : {full_name}")
```

```
10 + 20 = 30
  : 86.5
   :
```

3.2 Optional   Union

```python
from typing import Optional, Union

def find_student(students: List[str], name: str) -> Optional[int]:
    """              None """
    try:
        return students.index(name)
    except ValueError:
        return None

def process_number(value: Union[int, float, str]) -> float:
    """      float    """
    if isinstance(value, str):
        return float(value)
    return float(value)

#
students = ["  ", "  ", "  "]

index = find_student(students, "  ")
if index is not None:
    print(f"   {index}     ")
else:
    print("      ")

# Union
print(f"  42: {process_number(42)}")
print(f"    3.14: {process_number(3.14)}")
print(f"   '123.45': {process_number('123.45')}")
```

```
  1
 42: 42.0
   3.14: 3.14
  '123.45': 123.45
```

4

```python
from typing import ClassVar
from datetime import datetime

class Student:
    """   """

    #
    school_name: ClassVar[str] = "   "

    def __init__(self, name: str, age: int, grade: int) -> None:
        self.name: str = name
        self.age: int = age
        self.grade: int = grade
        self.scores: List[float] = []

    def add_score(self, score: float) -> None:
        """   """
        self.scores.append(score)

    def get_average_score(self) -> Optional[float]:
        """    """
        if not self.scores:
            return None
        return sum(self.scores) / len(self.scores)

    def get_info(self) -> Dict[str, Union[str, int, float, None]]:
        """      """
        return {
            "name": self.name,
            "age": self.age,
            "grade": self.grade,
            "average_score": self.get_average_score()
        }
    def __str__(self) -> str:
        return f" : {self.name} ({self.age} , {self.grade} )"

#
student = Student("  ", 20, 2)
student.add_score(85.5)
student.add_score(92.0)
student.add_score(78.5)

print(student)
print(f"  : {student.get_average_score():.1f}")
print(f"   : {student.get_info()}")
```

```
 :    (20 , 2 )
  : 85.3
   : {'name': '  ', 'age': 20, 'grade': 2, 'average_score': 85.33333333333333}
```

5

```python
from typing import TypeVar, Generic, List

#
T = TypeVar('T')

class Stack(Generic[T]):
    """        """

    def __init__(self) -> None:
        self._items: List[T] = []

    def push(self, item: T) -> None:
        """    """
        self._items.append(item)

    def pop(self) -> Optional[T]:
        """   """
        if self._items:
            return self._items.pop()
        return None

    def peek(self) -> Optional[T]:
        """     """
        if self._items:
            return self._items[-1]
        return None

    def is_empty(self) -> bool:
        """      """
        return len(self._items) == 0

    def size(self) -> int:
        """     """
        return len(self._items)
```

```
#
string_stack: Stack[str] = Stack()
string_stack.push(" ")
string_stack.push("  ")
string_stack.push("  ")

print(f"    : {string_stack.size()}")
print(f"   : {string_stack.peek()}")
print(f"  : {string_stack.pop()}")

#
number_stack: Stack[int] = Stack()
number_stack.push(10)
number_stack.push(20)
number_stack.push(30)

print(f"     : {number_stack.size()}")
print(f"   : {number_stack.peek()}")
```

```
    : 3
   :
  :
     : 3
   : 30
```

6

6.1 Callable

```python
from typing import Callable

def apply_operation(x: int, y: int, operation: Callable[[int, int], int]) -> int:
    """      """
    return operation(x, y)

def add(a: int, b: int) -> int:
    return a + b

def multiply(a: int, b: int) -> int:
    return a * b

#
result1 = apply_operation(10, 5, add)
result2 = apply_operation(10, 5, multiply)

print(f"10 + 5 = {result1}")
print(f"10 × 5 = {result2}")

#
result3 = apply_operation(10, 5, lambda x, y: x - y)
print(f"10 - 5 = {result3}")
```

```
10 + 5 = 15
10 × 5 = 50
10 - 5 = 5
```

6.2 Literal

```python
from typing import Literal

def set_color(color: Literal["red", "green", "blue"]) -> str:
    """       """
    return f"  {color}     "

def calculate_shape_area(shape: Literal["circle", "square", "triangle"], size: float) -> floa
    """      """
    import math

    if shape == "circle":
        return math.pi * (size / 2) ** 2
    elif shape == "square":
        return size ** 2
    elif shape == "triangle":
        return (size ** 2) * math.sqrt(3) / 4

    return 0.0

#
print(set_color("red"))
print(set_color("blue"))

area1 = calculate_shape_area("circle", 10)
area2 = calculate_shape_area("square", 10)
print(f"    ( 10): {area1:.2f}")
print(f"     ( 10): {area2:.2f}")
```

```
red
blue
 ( 10): 78.54
  ( 10): 100.00
```

7

7.1 mypy

```python
#    mypy

def divide_numbers(a: int, b: int) -> float:
    """         """
    if b == 0:
        raise ValueError("       ")
    return a / b

def process_user_data(data: Dict[str, Union[str, int]]) -> str:
    """      """
    name = data.get("name", " ")
    age = data.get("age", 0)

    if not isinstance(name, str):
        name = str(name)
    if not isinstance(age, int):
        age = int(age) if str(age).isdigit() else 0

    return f"{name} ({age} )"

#
result = divide_numbers(10, 3)
print(f"10 ÷ 3 = {result:.2f}")

user = {"name": "   ", "age": 25}
info = process_user_data(user)
print(f"    : {info}")
```

```
10 ÷ 3 = 3.33
   :    (25 )
```

8

```python
from typing import NamedTuple
from enum import Enum
from datetime import datetime, date

class Priority(Enum):
    """     """
    LOW = 1
    MEDIUM = 2
    HIGH = 3
    URGENT = 4

class Task(NamedTuple):
    """      """
    id: int
    title: str
    description: str
    priority: Priority
    due_date: Optional[date]
    completed: bool = False

class TaskManager:
    """       """

    def __init__(self) -> None:
        self._tasks: Dict[int, Task] = {}
        self._next_id: int = 1

    def create_task(
        self,
        title: str,
        description: str,
        priority: Priority = Priority.MEDIUM,
        due_date: Optional[date] = None
    ) -> Task:
        """     """
        task = Task(
            id=self._next_id,
            title=title,
            description=description,
            priority=priority,
            due_date=due_date
        )
        self._tasks[self._next_id] = task
        self._next_id += 1
        return task

    def get_task(self, task_id: int) -> Optional[Task]:
        """ID    """
        return self._tasks.get(task_id)

    def get_tasks_by_priority(self, priority: Priority) -> List[Task]:
        """       """
        return [task for task in self._tasks.values() if task.priority == priority]

    def complete_task(self, task_id: int) -> bool:
        """      """
        if task_id in self._tasks:
            old_task = self._tasks[task_id]
            self._tasks[task_id] = old_task._replace(completed=True)
            return True
        return False

    def get_overdue_tasks(self) -> List[Task]:
        """      """
        today = date.today()
        return [
            task for task in self._tasks.values()
            if task.due_date and task.due_date < today and not task.completed
        ]

#
manager = TaskManager()

#
task1 = manager.create_task(
    "   ",
    "Python          ",
    Priority.HIGH,
    date(2024, 12, 31)
)

task2 = manager.create_task(
    "  ",
    "     ",
    Priority.MEDIUM
)

task3 = manager.create_task(
    "   ",
    "       ",
    Priority.URGENT,
    date(2024, 1, 15)
)

print(f"    : {task1}")
print(f"    : {len(manager.get_tasks_by_priority(Priority.HIGH))}")

#
success = manager.complete_task(task2.id)
print(f"   : {success}")

#
overdue = manager.get_overdue_tasks()
print(f"    : {len(overdue)}")
```

```
    : Task(id=1, title='    ', description='Python          ', priority=<Priority.HIGH: 3>, du
    : 1
   : True
    : 2
```

9

9.1 1.

```
#
def important_function(data: List[str]) -> Dict[str, int]:
    """      """
    result = {}
    for item in data:
        result[item] = len(item)
    return result

#
```

9.2 2.

```
from typing import TypeAlias, Any

#
UserData: TypeAlias = Dict[str, Union[str, int, List[str]]]
ProcessResult: TypeAlias = Tuple[bool, str, Optional[Dict[str, Any]]]

def process_user(data: UserData) -> ProcessResult:
    """      """
    try:
        #
        return True, "  ", {"processed": True}
    except Exception as e:
        return False, str(e), None
```

10

1. :
2. **IDE** :
3. :
4. :
5. :

10.1

-
-
- **mypy pyright**
- **TypeAlias**
-

Python Python

11