Python

Table of contents

1

1.1

- **Python 3.5+**
-
-
- **IDE**, ,
-

```
#
def add(a, b):
    return a + b

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

#
result: int = add(5, 3)  #
```

1.2

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

#
def greet(name: str) -> str:
    return f"   {name}  "
```

```python
def calculate_area(radius: float) -> float:
    return 3.14159 * radius ** 2

def print_info(name: str, age: int) -> None:  #
    print(f"{name}  {age}  ")

#
message: str = greet(" ")
area: float = calculate_area(5.0)
print_info(" ", 30)
```

1.3

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

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

#
student: Dict[str, int] = {" ": 85, " ": 92}
config: Dict[str, str] = {"host": "localhost", "port": "8080"}

#
point: Tuple[int, int] = (10, 20)
person: Tuple[str, int, bool] = (" ", 25, True)

#
unique_numbers: Set[int] = {1, 2, 3, 4, 5}

#     None
maybe_name: Optional[str] = None  # str | None
maybe_age: Optional[int] = 25

def find_user(user_id: int) -> Optional[str]:
    """         None  """
    users = {1: " ", 2: " "}
    return users.get(user_id)
```

```
#
user: Optional[str] = find_user(1)  # " "
unknown: Optional[str] = find_user(99)  # None
```

1.4

```
from typing import Callable, List, Any

#
def apply_operation(numbers: List[int], operation: Callable[[int], int]) -> List[int]:
    """      """
    return [operation(num) for num in numbers]

def square(x: int) -> int:
    return x ** 2

def double(x: int) -> int:
    return x * 2

#
numbers = [1, 2, 3, 4, 5]
squared = apply_operation(numbers, square)    # [1, 4, 9, 16, 25]
doubled = apply_operation(numbers, double)    # [2, 4, 6, 8, 10]

#
Calculator = Callable[[float, float], float]

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

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

def perform_calculation(calc: Calculator, x: float, y: float) -> float:
    return calc(x, y)

result1 = perform_calculation(add, 5.0, 3.0)       # 8.0
result2 = perform_calculation(multiply, 4.0, 2.0)  # 8.0
```

1.5

```python
from typing import List, Optional, ClassVar
from dataclasses import dataclass

class Student:
    #
    school_name: ClassVar[str] = "Python "
    total_students: ClassVar[int] = 0

    def __init__(self, name: str, student_id: str, grades: Optional[List[int]] = None):
        self.name: str = name
        self.student_id: str = student_id
        self.grades: List[int] = grades or []
        Student.total_students += 1

    def add_grade(self, grade: int) -> None:
        self.grades.append(grade)

    def get_average(self) -> Optional[float]:
        if not self.grades:
            return None
        return sum(self.grades) / len(self.grades)

    def __str__(self) -> str:
        return f"Student({self.name}, {self.student_id})"

#
@dataclass
class Point:
    x: float
    y: float

    def distance_from_origin(self) -> float:
        return (self.x ** 2 + self.y ** 2) ** 0.5

@dataclass
class Course:
    name: str
    credits: int
    students: List[Student]
```

```
    instructor: Optional[str] = None

    def add_student(self, student: Student) -> None:
        self.students.append(student)

    def get_enrollment_count(self) -> int:
        return len(self.students)

#
student1 = Student(" ", "S001", [85, 92, 78])
student2 = Student(" ", "S002")

course = Course("Python ", 3, [student1])
course.add_student(student2)

point = Point(3.0, 4.0)
distance: float = point.distance_from_origin()  # 5.0
```

1.6 Union Python 3.10+

```
from typing import Union, List, Dict

# Union
def process_id(user_id: Union[int, str]) -> str:
    return str(user_id).upper()

# Python 3.10+
def process_id_new(user_id: int | str) -> str:
    return str(user_id).upper()

#  Union
ApiResponse = Union[Dict[str, str], List[Dict[str, str]], str]

def handle_api_response(response: ApiResponse) -> None:
    if isinstance(response, dict):
        print(f"    : {response}")
    elif isinstance(response, list):
        print(f"    : {len(response)}   ")
    else:
```

```
        print(f"    : {response}")

# Optional  Union[T, None]
def find_by_name(name: str) -> Optional[Student]:
    # Python 3.10+  Student | None
    pass

#
process_id(123)        # "123"
process_id("abc")      # "ABC"

handle_api_response({"status": "success"})
handle_api_response([{"id": 1}, {"id": 2}])
handle_api_response("     ")
```

1.7

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

#
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("Hello")
string_stack.push("World")
top_string: Optional[str] = string_stack.pop()  # "World"

#
int_stack: Stack[int] = Stack()
int_stack.push(1)
int_stack.push(2)
top_int: Optional[int] = int_stack.pop()  # 2

#
def get_first_item(items: List[T]) -> Optional[T]:
    return items[0] if items else None

first_str: Optional[str] = get_first_item(["a", "b", "c"])  # "a"
first_int: Optional[int] = get_first_item([1, 2, 3])       # 1
```

1.8

```
# mypy

def calculate_discount(price: float, discount_rate: float) -> float:
    if discount_rate < 0 or discount_rate > 1:
        raise ValueError("  0-1      ")
    return price * (1 - discount_rate)

#
final_price: float = calculate_discount(100.0, 0.1)  # OK
```

```
#    mypy
# final_price = calculate_discount("100", "0.1")  # error: Argument 1 has incompatible type
# final_price = calculate_discount(100.0)         # error: Too few arguments

# pyright    pyproject.toml
"""
[tool.pyright]
include = ["src"]
exclude = ["**/__pycache__"]
reportMissingImports = true
reportMissingTypeStubs = false
pythonVersion = "3.12"
"""

# mypy    mypy.ini
"""
[mypy]
python_version = 3.12
warn_return_any = True
warn_unused_configs = True
disallow_untyped_defs = True
"""
```

1.9

1.10 :

```
from typing import List, Dict, Optional, Union, Callable, TypeVar, Generic
from dataclasses import dataclass
from datetime import datetime

#
UserId = int
BookId = str
ISBN = str

@dataclass
class User:
    id: UserId
```

```
    name: str
    email: str
    registration_date: datetime

    def __str__(self) -> str:
        return f"User(id={self.id}, name='{self.name}')"

@dataclass
class Book:
    id: BookId
    title: str
    author: str
    isbn: ISBN
    available_copies: int

    def is_available(self) -> bool:
        return self.available_copies > 0

    def __str__(self) -> str:
        return f"Book('{self.title}' by {self.author})"

@dataclass
class BorrowRecord:
    user_id: UserId
    book_id: BookId
    borrow_date: datetime
    return_date: Optional[datetime] = None

    def is_returned(self) -> bool:
        return self.return_date is not None

class LibraryError(Exception):
    """     """
    pass

class Library:
    def __init__(self, name: str) -> None:
        self.name: str = name
        self.users: Dict[UserId, User] = {}
        self.books: Dict[BookId, Book] = {}
        self.borrow_records: List[BorrowRecord] = []
        self._next_user_id: UserId = 1
```

```
    def register_user(self, name: str, email: str) -> User:
        """    """
        user = User(
            id=self._next_user_id,
            name=name,
            email=email,
            registration_date=datetime.now()
        )
        self.users[user.id] = user
        self._next_user_id += 1
        return user

    def add_book(self, title: str, author: str, isbn: ISBN, copies: int = 1) -> Book:
        """   """
        book_id = f"B{len(self.books) + 1:04d}"
        book = Book(
            id=book_id,
            title=title,
            author=author,
            isbn=isbn,
            available_copies=copies
        )
        self.books[book_id] = book
        return book

    def find_user(self, user_id: UserId) -> Optional[User]:
        """    """
        return self.users.get(user_id)

    def find_book(self, book_id: BookId) -> Optional[Book]:
        """   """
        return self.books.get(book_id)

    def search_books(self, query: str) -> List[Book]:
        """         """
        query_lower = query.lower()
        results: List[Book] = []

        for book in self.books.values():
            if (query_lower in book.title.lower() or
                query_lower in book.author.lower()):
                results.append(book)
```

```
        return results

    def borrow_book(self, user_id: UserId, book_id: BookId) -> BorrowRecord:
        """    """
        user = self.find_user(user_id)
        if not user:
            raise LibraryError(f"  ID {user_id}      ")

        book = self.find_book(book_id)
        if not book:
            raise LibraryError(f" ID {book_id}      ")

        if not book.is_available():
            raise LibraryError(f"'{book.title}'      ")

        #
        record = BorrowRecord(
            user_id=user_id,
            book_id=book_id,
            borrow_date=datetime.now()
        )

        #
        book.available_copies -= 1
        self.borrow_records.append(record)

        return record

    def return_book(self, user_id: UserId, book_id: BookId) -> bool:
        """   """
        #
        for record in self.borrow_records:
            if (record.user_id == user_id and
                record.book_id == book_id and
                not record.is_returned()):

                #
                record.return_date = datetime.now()
                book = self.find_book(book_id)
                if book:
                    book.available_copies += 1
                return True
```

```
        return False

    def get_user_borrowed_books(self, user_id: UserId) -> List[Book]:
        """       """
        borrowed_books: List[Book] = []

        for record in self.borrow_records:
            if record.user_id == user_id and not record.is_returned():
                book = self.find_book(record.book_id)
                if book:
                    borrowed_books.append(book)

        return borrowed_books

    def get_overdue_books(self, days: int = 14) -> List[tuple[User, Book, int]]:
        """     """
        from datetime import timedelta

        overdue: List[tuple[User, Book, int]] = []
        cutoff_date = datetime.now() - timedelta(days=days)

        for record in self.borrow_records:
            if not record.is_returned() and record.borrow_date < cutoff_date:
                user = self.find_user(record.user_id)
                book = self.find_book(record.book_id)
                if user and book:
                    days_overdue = (datetime.now() - record.borrow_date).days
                    overdue.append((user, book, days_overdue))

        return overdue

#
def main() -> None:
    #
    library = Library("   ")

    #
    user1 = library.register_user("  ", "tanaka@example.com")
    user2 = library.register_user("  ", "sato@example.com")

    #
    book1 = library.add_book("Python ", "  ", "978-1234567890", 3)
```

```
    book2 = library.add_book("   ", "   ", "978-0987654321", 2)
    book3 = library.add_book("   ", "   ", "978-1111111111", 1)

    #
    search_results: List[Book] = library.search_books("Python")
    print(f"   : {len(search_results)} ")

    #
    try:
        record1 = library.borrow_book(user1.id, book1.id)
        record2 = library.borrow_book(user2.id, book2.id)
        print("    ")
    except LibraryError as e:
        print(f"  : {e}")

    #
    borrowed: List[Book] = library.get_user_borrowed_books(user1.id)
    print(f"{user1.name}   : {len(borrowed)} ")

    #
    success: bool = library.return_book(user1.id, book1.id)
    print(f"   : {success}")

if __name__ == "__main__":
    main()
```

1.11

1. -
2. **API** -
3. - mypy, pyright
4. **Union** -
5. -

1.12

- (async/await)
- ( )
- (NumPy, Pandas)

- (Web)

1.13

1.13.1

: - Python.org - - mypy - Real Python -

1.14

:

:
|

: