Type Hinting

Annotate code with expected types of variables & function

vedicskill.

Why Type Hinting Exits

why we need to annotate the code with expected data type ?

→ What type of value is expected here ?

→ Can this variable ever be None ?

→ What exactly does this function return ?

***Type hints** acts like a documentation for humans to read and never goes out of sync with the code*

mypy

How Python Works on “type” of variable

```
>>> x = 10
```

We do not tell the language what type a variable is.

I can write,

```
>>> x = “ten”
```

Python will accept without an error.

Type hints *exits to capture variable intention. They are for humans and tools not for python*

vedicskill.

Next

Hands on Type Hinting

```
import typing
variable: str = “Hello world”
```

vedicskill.

Type Hinting in Python

```
import typing
variable: str = “Hello world”
```

Type Hints in Python

Type hints are feature in Python that allow developers to annotate their code with expected type for variables and function arguments.

```python
variable = value


variable: type = value
```

int, float, str, List,
Dict, Tuple, …

Type Hints in Python

Type hints are feature in Python that allow developers to annotate their code with expected type for variables and function arguments.

```python
age: int = 25


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

VEDICSKILL.

Type hint for Collection

`List, Set, Tuple, Dict`

typing module

Provides type hints, to annotate variables, functions parameters, return types

The typing module

Python introduction the typing module for other types of utilities

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

l: List[int] = [1, 2, 3, 4, 5]

t: Tuple[int, str, float] = (1, "hello", 3.14)

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

d: Dict[str, int] = {"a": 1, "b": 2, "c": 3}
```

Union

Specify multiple data types

vedicskill.

Union

For having a list with different elements of different types, Union helps to combine the different types

```python
from typing import List, Union

l: List[Union[int, float]] = [1, 2.5, 3.14, 5]
```

vedicskill.

Optional

Union[…,None]

vedicskill.

Optional

If the types hints or arguments that are not mandatory, then Optional in typing is generally use. Where they can be omitted or potentially have a value of None

python

```python
from typing import Optional

def greeting(name: Optional[str] = None) -> str:
       return f"Hello, {name if name else 'Anonymous'}"
```

VEDICSKILL.

Any

Skip Type Checking for that variable or function.

Any

In some situations, the code is so dynamic or complicated that it won’t be possible to annotate it correctly. Any type will be useful in this situation.

```python
from typing import Any

def f(x: Any) -> Any:
    return x
```

Callable

When you pass function as argument of other function, then `Callable` in typing is generally use. It can be useful to have types for function signatures.

```python
from typing import Callable

def run(func: Callable[[int],int], x: int) -> str:
        number_type: str = "Odd" if fun(x) == 1 else "Even“
        return number_type


def remainder(x : int) -> int:
        value: int = x%2
        return value
```

VEDICSKILL.

Next

Pydantic

Data Validation on Runtime

vedicskill.