



马哥教育

IT 人的高薪职业学院

# Python类型注解

讲师：Wayne

从业十余载，漫漫求知路

# 函数定义的弊端

- ❑ Python是动态语言，变量随时可以被赋值，且能赋值为不同的类型
- ❑ Python不是静态编译型语言，变量类型是在运行器决定的
- ❑ 动态语言很灵活，但是这种特性也是弊端

```
def add(x, y):  
    return x + y  
print(add(4, 5))  
print(add('hello', 'world'))  
add(4, 'hello') #
```

- ❑ 难发现：由于不做任何类型检查，直到运行期问题才显现出来，或者线上运行时才能暴露出问题
- ❑ 难使用：函数的使用者看到函数的时候，并不知道你的函数的设计，并不知道应该传入什么类型的数据



# 函数定义的弊端

□ 如何解决这种动态语言定义的弊端呢？

□ 增加文档Documentation String

□ 这只是一个惯例，不是强制标准，不能要求程序员一定为函数提供说明文档

□ 函数定义更新了，文档未必同步更新

```
def add(x, y):  
    ...  
  
    :param x: int  
    :param y: int  
    :return: int  
    ...  
  
    return x + y  
print(help(add))
```



# 函数注解Function Annotations

□ 如果解决这种动态语言定义的弊端呢？

□ 函数注解

```
def add(x:int , y:int) -> int :
```

```
    ...
```

```
    :param x: int
```

```
    :param y: int
```

```
    :return: int
```

```
    ...
```

```
    return x + y
```

```
print(help(add))
```

```
print(add(4, 5))
```

```
print(add('mag', 'edu'))
```



# 函数注解Function Annotations

## □ 函数注解

- Python 3.5引入
- 对函数的参数进行类型注解
- 对函数的返回值进行类型注解
- 只对函数参数做一个辅助的说明，并不对函数参数进行类型检查
- 提供给第三方工具，做代码分析，发现隐藏的bug
- 函数注解的信息，保存在\_\_annotations\_\_属性中

```
add.__annotations__  
{'x': <class 'int'>, 'y': <class 'int'>, 'return': <class 'int'>}
```

## □ 变量注解

- Python 3.6引入

```
i : int = 3
```

# 业务应用

## □ 函数参数类型检查

### □ 思路

- 函数参数的检查，一定是在函数外
- 函数应该作为参数，传入到检查函数中
- 检查函数拿到函数传入的实际参数，与形参声明对比
- `__annotations__` 属性是一个字典，其中包括返回值类型的声明。假设要做位置参数的判断，无法和字典中的声明对应。使用inspect模块

### □ inspect模块

- 提供获取对象信息的函数，可以检查函数和类、类型检查



# inspect模块

- `signature(callable)` , 获取签名 ( 函数签名包含了一个函数的信息 , 包括函数名、它的参数类型、它所在的类和名称空间及其他信息 )

```
import inspect
def add(x:int, y:int, *args,**kwargs) -> int:
    return x + y
sig = inspect.signature(add)
print(sig, type(sig)) # 函数签名
print('params : ', sig.parameters) # OrderedDict
print('return : ', sig.return_annotation)
print(sig.parameters['y'], type(sig.parameters['y']))
print(sig.parameters['x'].annotation)
print(sig.parameters['args'])
print(sig.parameters['args'].annotation)
print(sig.parameters['kwargs'])
print(sig.parameters['kwargs'].annotation)
```

# inspect模块

- ❑ `inspect.isfunction(add)`，是否是函数
- ❑ `inspect.ismethod(add)`，是否是类的方法
- ❑ `inspect.isgenerator(add)`，是否是生成器对象
- ❑ `inspect.isgeneratorfunction(add)`，是否是生成器函数
- ❑ `inspect.isclass(add)`，是否是类
- ❑ `inspect.ismodule(inspect)`，是否是模块
- ❑ `inspect.isbuiltin(print)`，是否是内建对象
- ❑ 还有很多is函数，需要的时候查阅inspect模块帮助



# inspect模块

## □ Parameter对象

- 保存在元组中，是只读的
- name，参数的名字
- annotation，参数的注解，可能没有定义
- default，参数的缺省值，可能没有定义
- empty，特殊的类，用来标记default属性或者注释annotation属性的空值
- kind，实参如何绑定到形参，就是形参的类型
  - POSITIONAL\_ONLY，值必须是位置参数提供
  - POSITIONAL\_OR\_KEYWORD，值可以作为关键字或者位置参数提供
  - VAR\_POSITIONAL，可变位置参数，对应\*args
  - KEYWORD\_ONLY，keyword-only参数，对应\*或者\*args之后的出现的非可变关键字参数
  - VAR\_KEYWORD，可变关键字参数，对应\*\*kwargs

# inspect模块

## □ 举例

```
import inspect

def add(x, y:int=7, *args, z, t=10,**kwargs) -> int:
    return x + y
sig = inspect.signature(add)
print(sig)
print('params : ', sig.parameters) # 有序字典
print('return : ', sig.return_annotation)
print('~~~~~')
for i, item in enumerate(sig.parameters.items()):
    name, param = item
    print(i+1, name, param.annotation, param.kind, param.default)
    print(param.default is param.empty, end='\n\n')
```

# 业务应用

- 有函数如下

```
def add(x, y:int=7) -> int:  
    return x + y
```

- 请检查用户输入是否符合参数注解的要求？





# 业务应用

- 有函数如下

```
def add(x, y:int=7) -> int:  
    return x + y
```

- 请检查用户输入是否符合参数注解的要求？

- 思路

- 调用时，判断用户输入的实参是否符合要求
- 调用时，用户感觉上还是在调用add函数
- 对用户输入的数据和声明的类型进行对比，如果不符合，提示用户

# 业务应用

```
import inspect
def add(x, y:int=7) -> int:
    return x + y
```

```
def check(fn):
    def wrapper(*args, **kwargs):
        sig = inspect.signature(fn)
        params = sig.parameters
        values = list(params.values())
        for i,p in enumerate(args):
            if isinstance(p, values[i].annotation): # 实参和形参声明一致
                print('==')
        for k,v in kwargs.items():
            if isinstance(v, params[k].annotation): # 实参和形参声明一致
                print('===')
        return fn(*args, **kwargs)
    return wrapper
```

调用测试

```
check(add)(20,10)
check(add)(20,y=10)
check(add)(y=10,x=20)
```

业务需求是参数有注解就要求实参类型和声明应该一致，没有注解的参数不比较，如何修改代码？

# 业务应用

```
import inspect
```

```
def check(fn):  
    def wrapper(*args, **kwargs):  
        sig = inspect.signature(fn)  
        params = sig.parameters  
        values = list(params.values())  
        for i,p in enumerate(args):  
            param = values[i]  
            if param.annotation is not param.empty and not isinstance(p, param.annotation):  
                print(p,'!=',values[i].annotation)  
        for k,v in kwargs.items():  
            if params[k].annotation is not inspect.empty and not isinstance(v, params[k].annotation):  
                print(k,v,'!=',params[k].annotation)  
        return fn(*args, **kwargs)  
    return wrapper
```

```
@check
```

```
def add(x, y:int=7) -> int:  
    return x + y
```

```
调用测试  
add(20,10)  
add(20,y=10)  
add(y=10,x=20)
```



马哥教育  
IT人的高薪职业学院

# 谢谢

咨询热线 400-080-6560

