Задача 3:  
1. Функция 1:

def func1():

param = 4

def inner():

param += 1

return param

return param

В этом примере произойдет ошибка, так как внутри функции inner() происходит попытка изменить значение переменной param, которая является локальной для внешней функции func1(). Переменная param внутри inner() не была объявлена, что приводит к ошибке UnboundLocalError: local variable 'param' referenced before assignment. Для исправления данной ошибки необходимо использовать ключевое слово nonlocal, чтобы указать, что изменение должно происходить с переменной из внешней области видимости. 2. Функция 2:

def func2():

param = 4

def inner(var):

var += 1

return var

inner(param)

return param

В данном случае все пройдет без ошибок. Функция inner() получает копию значения param из области видимости func2(), затем увеличивает ее на 1 и возвращает новое значение. Однако при этом значение param во внешней функции func2() не изменяется, поэтому она вернет первоначальное значение 4. 3. Функция 3:

def func3():

param = 4

def inner(var):

var += 1

return var

param = inner(param)

return param

В этом примере все также пройдет без ошибок. Функция inner() получает значение param из области видимости func3(), увеличивает его на 1 и возвращает новое значение. Это новое значение затем присваивается переменной param во внешней функции func3(), которая и возвращается в итоге. Таким образом, можно сделать следующие выводы: 1. Функция 1 вызовет ошибку UnboundLocalError, так как внутри inner() происходит попытка изменить локальную для func1() переменную param. 2. Функция 2 вернет исходное значение param, так как изменения происходят только с копией внутри inner(). 3. Функция 3 вернет новое значение param, так как значение обновляется во внешней функции.

Задача 5:

Для реализации функции inspect(), которая анализирует другую функцию и выводит информацию о ее параметрах, можно использовать модуль inspect в Python. Вот пример реализации:

import inspect

def inspect(func):

"""

Функция, которая анализирует другую функцию и выводит информацию о ее параметрах.

Args:

func (callable): Функция, которую необходимо проанализировать.

"""

print(f"Анализируемая функция: {func.\_\_name\_\_}")

# Получаем информацию о параметрах функции

signature = inspect.signature(func)

# Итерируемся по параметрам и выводим их название и тип

for param in signature.parameters.values():

param\_type = param.annotation if param.annotation != inspect.Parameter.empty else "Не указан"

print(f"Параметр: {param.name} | Тип: {param\_type}")

# Пример использования

def example\_function(a: int, b: str, c: float = 3.14, \*args, d: bool, \*\*kwargs):

pass

inspect(example\_function)

Вывод:

Анализируемая функция: example\_function

Параметр: a | Тип: <class 'int'>

Параметр: b | Тип: <class 'str'>

Параметр: c | Тип: <class 'float'>

Параметр: args | Тип: Не указан

Параметр: d | Тип: <class 'bool'>

Параметр: kwargs | Тип: Не указан

Объяснение: 1. Функция inspect() принимает в качестве аргумента другую функцию, которую необходимо проанализировать. 2. Внутри inspect() используется функция inspect.signature(func) для получения информации о параметрах анализируемой функции. 3. Затем мы итерируемся по параметрам, используя signature.parameters.values(), и выводим название каждого параметра и его тип. 4. Если тип параметра не указан явно (аннотацией), то выводится "Не указан". 5. В примере используется функция example\_function(), которая имеет различные типы параметров (позиционные, ключевые, переменное количество аргументов). Таким образом, функция inspect() успешно анализирует переданную ей функцию и выводит информацию о ее параметрах, включая их названия и типы.