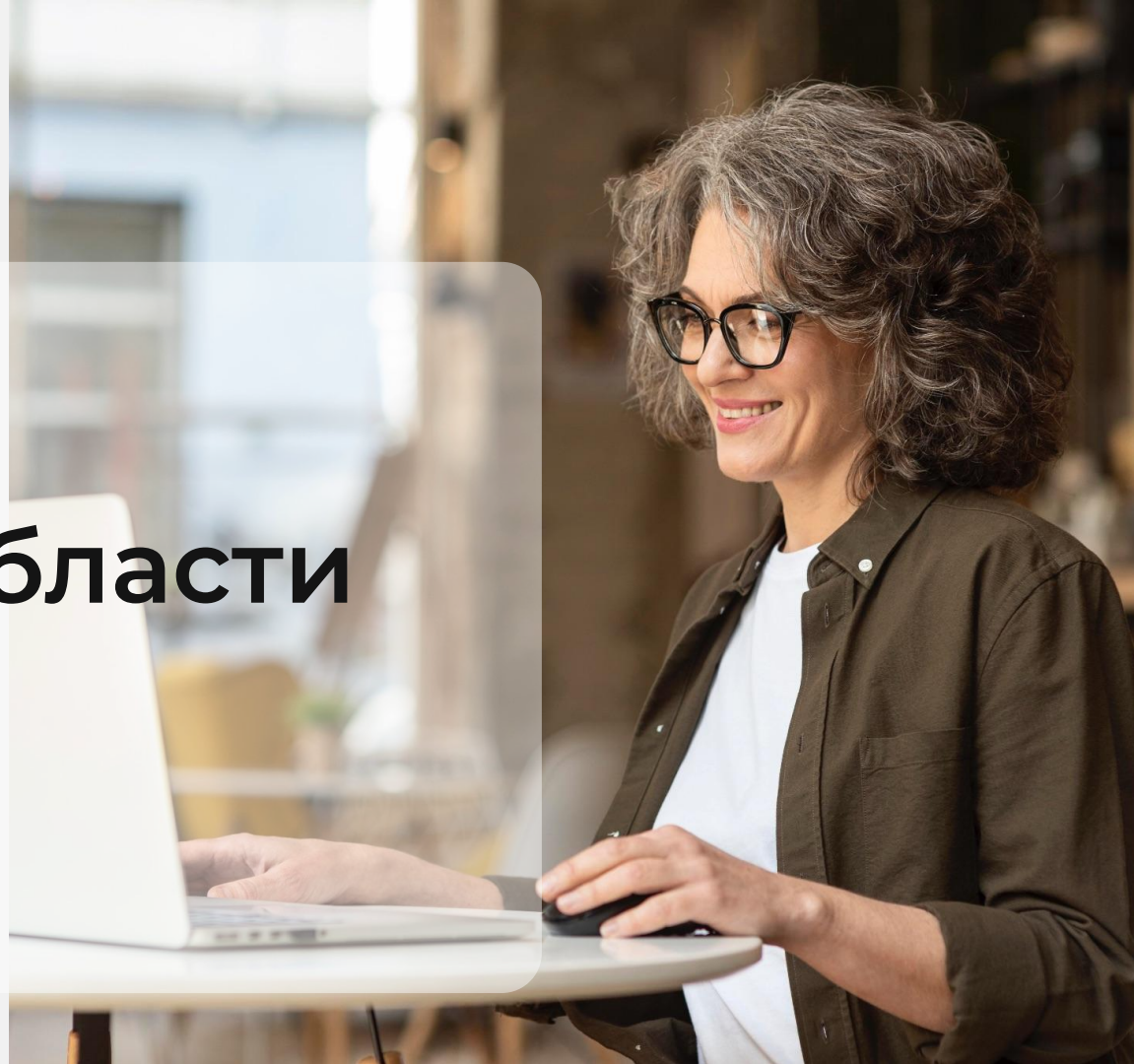
IT CAREER HUB

Python

Функции. Области видимости



Преподаватель

Портрет

Имя Фамилия

Текущая должность

Количество лет опыта

Какой у Вас опыт - ключевые кейсы

Самые яркие проекты

Дополнительная информация по вашему усмотрению

Корпоративный e-mail

Социальные сети (по желанию)

Важно

- Камера должна быть включена на протяжении всего занятия
- В течение занятия вопросы задавать в чате или когда преподаватель спрашивает, есть ли у Вас вопросы
- Вести себя уважительно и этично по отношению к остальным участникам занятия
- Организационные вопросы по обучению решаются с кураторами, а не на тематических занятиях
- Во время занятия будут интерактивные задания, будьте готовы включить камеру или демонстрацию экрана по просьбе преподавателя

План занятия

- Функция
- Ключевое слово pass
- Ключевое слово def
- Ключевое слово pass
- Ключевое слово def
- Вызов функции
- Аргументы функций
- Комбинация различных типов аргументов

ICH IT CAREER HUB

ОСНОВНОЙ БЛОК

IT CAREER HUB

Функция

Функция

Именованный блок кода, предназначенный для выполнения определённой задачи. Функции позволяют переиспользовать код, делая программы более организованными и читаемыми.

Функция

Ранее мы использовали только готовые функции. На этом занятии мы научимся создавать функции самостоятельно.

Зачем нужны функции?

- **Повторное использование кода**: Один раз написав функцию, её можно вызывать многократно.
- **Читаемость**: Функции помогают разбивать программу на логические части.
- **Модульность**: Удобно структурировать код, разбивая его на независимые части.
- **Упрощение отладки**: Изменения в функции автоматически применяются во всех местах её вызова.

Синтаксис

```
def function_name(parameters):
    # Тело функции: код, выполняющийся при вызове
    return result  # Возвращаемое значение (опционально)
```

Зачем нужны функции?

- **def**: ключевое слово, обозначающее создание функции.
- **function_name**: имя функции, используемое для её вызова.
- **parameters**: входные данные (аргументы), которые передаются в функцию.
- **return**: возвращает результат выполнения функции (необязательно).

Синтаксис

```
def greet(name):
    print(f"Привет, {name}!")

greet("Алиса")
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

Ключевое слово pass

Ключевое слово pass

Ключевое слово **pass** используется как заглушка. Оно позволяет определить пустой блок кода, который не выполняет никаких действий. Это полезно, когда вы планируете добавить код позже, но хотите сохранить корректный синтаксис.

Особенности

- **Ничего не делает**: `pass` ничего не выполняет, он просто позволяет избежать ошибок синтаксиса в местах, где блок кода обязателен.
- **Часто используется для заглушек**: Позволяет временно оставить место под код, который будет добавлен позже.
- **Применяется в конструкциях с обязательным блоком кода**: Например, в циклах, функциях, условиях.

Примеры

```
#                                                                    вие
if                                                                   ue:
    pass  # Блок кода будет добавлен позже

else:
    print("False")



#                                                                    Цикл
for                          i                          in           range(5):
    pass  # Цикл ничего не делает, но синтаксически корректен



#                                                                    Функция
def                                                                  validate_data():
    pass  # Функция пока не реализована
```

IT CAREER HUB

ВОПРОСЫ

ICH IT CAREER HUB

Ключевое слово def

Полезно знать

def

Это ключевое слово, которое используется для определения пользовательской функции.

Полезно знать

ICH
IT CAREER HUB

Имя функции

Это уникальный идентификатор, используемый для её вызова. Оно должно быть информативным и описывать, что делает функция.

Правила именования функций

- Имя должно начинаться с буквы или символа `_`, но не с цифры.
- Нельзя использовать зарезервированные слова Python (например, `def`, `return`, `if`).
- Не рекомендуется переопределять встроенные функции (`print`, `sum`).
- Следуйте соглашению PEP 8: используйте `snake_case`.
- Используйте глаголы и информативные имена:

Примеры

```
def calculate_total():  # Хорошо
    pass

def total():  # Плохо, не указывает на действие
    pass
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

Вызов функции

Полезно знать

ICH
IT CAREER HUB

Вызов функции

Это процесс выполнения ранее определённого блока кода (функции) с помощью её имени. При вызове можно передать функции необходимые аргументы и получить результат её работы.

Примеры

```python
def greet():
    print("Hello!")


# Вызов функции без аргументов
greet()
```

Примеры

```python
def greet_person(name):
    print(f"Hello, {name}!")

# Вызов функции с аргументами
greet_person("Alice")
```

IT CAREER HUB

ВОПРОСЫ

ICH IT CAREER HUB

Аргументы функций

Полезно знать

ICH
IT CAREER HUB

Аргументы функций

Это значения, которые передаются функции при её вызове. С их помощью можно передавать данные, которые функция использует для выполнения своих задач.

Типы аргументов функций

- Передаются функции в порядке, указанном в определении.
- **Порядок важен**: каждое значение будет присвоено соответствующему параметру.
- **Количество ожидаемых аргументов должно совпадать с количеством переданных аргументов.**

Примеры

```python
def greet(name, age):  # Ожидаемые аргументы
    print(f"My name is {name} and I am {age} years old.")


greet("Alice", 25)  # Переданные аргументы
greet("Bob", 30)  # Переданные аргументы
```

Ключевое слово pass

Значения `"Alice"` и `25` передаются в переменные `name` и `age` соответственно. Теперь переменные `name` и `age` можно использовать внутри функции. При каждом вызове переменные принимают новые значения.

Полезно знать

ICH
IT CAREER HUB

TypeError

Это исключение, которое возникает, когда операция или функция применяется к объекту неподходящего типа. Эта ошибка сигнализирует, что программа пытается выполнить действие, которое не поддерживается данным типом данных.

TypeError

Например, TypeError возникает если количество переданных аргументов не совпадает с количеством ожидаемых аргументов.

Передано меньше аргументов, чем ожидается

```python
def greet(name, age):  # Ожидаемые аргументы
    print(f"My name is {name} and I am {age} years old.")

greet("Alice")  # Ошибка: меньше чем ожидается
```

Передано больше аргументов, чем ожидается

Если передать больше аргументов, чем указано в определении функции, Python сообщит об избыточных аргументах.

Передано больше аргументов, чем ожидается

```
def greet(name, age):  # Ожидаемые аргументы
   print(f"My name is {name} and I am {age} years old.")

greet("Alice", 25, "Minsk")  # Ошибка: больше чем ожидается
```

Именованные аргументы

Передаются с указанием имени параметра, что делает вызов функции более понятным. Порядок следования не имеет значения.

Пример

```python
def greet(name, age):  # Ожидаемые аргументы
    print(f"My name is {name} and I am {age} years old.")

greet(age=30, name="Bob")  # Именованные аргументы
```

Аргументы по умолчанию

При определении функции можно указать значения по умолчанию для аргументов. Если аргумент не передан, используется значение по умолчанию. Сначала указываются обязательные аргументы (без значения по умолчанию), а затем — аргументы со значениями по умолчанию. Нарушение этого порядка приведёт к ошибке синтаксиса.

Пример

```python
def greet(name, greeting="Hello"):
    print(f"{greeting}, {name}!")


greet("Alice")  # Приветствие не передано, будет использовано "Hello"
greet("Bob", "Hi")  # Вывод: Hi, Bob!
```

Полезно знать

Упаковка аргументов

Это процесс, при котором передаваемые в функцию аргументы объединяются в одну коллекцию.

Упаковка аргументов в функции

- кортеж для позиционных аргументов (с помощью символа *)
- словарь для именованных аргументов (с помощью символов **).

Полезно знать

ICH
IT CAREER HUB

Параметры *args и **kwargs

Это специальные параметры, которые позволяют передавать в функцию переменное количество аргументов.

Полезно знать

ICH
IT CAREER HUB

*args

Этот параметр позволяет передавать любое количество позиционных аргументов (в том числе ни одного). Аргументы упаковываются в кортеж.

Пример

```
def calculate_sum(*args):
    print("Аргументы:", args)
    print("Сумма:", sum(args))


calculate_sum(1, 2, 3)
calculate_sum()
```

Полезно знать

IT CAREER HUB

**kwargs

Этот параметр позволяет передавать любое количество именованных аргументов. Аргументы упаковываются в словарь.

Пример

```python
def print_user_info(**kwargs):
    print("Информация о пользователе:")
    for key, value in kwargs.items():
        print(f"\t{key}: {value}")


print_user_info(name="Alice", age=25, city="New York")
print_user_info()
```

IT CAREER HUB

ВОПРОСЫ

ICH IT CAREER HUB

Комбинация различных типов аргументов

Комбинация различных типов аргументов

В одной функции можно использовать разные типы аргументов. При этом важно соблюдать их порядок.

Порядок

- Позиционные аргументы
- `*args`
- Аргументы по умолчанию
- `**kwargs`

Пример

```python
def show_full_info(name, *args, age=25, **kwargs):
    print(f"Name: {name}")
    print(f"Other details: {args}")
    print(f"Age: {age}")
    print(f"Additional info: {kwargs}")

show_full_info("Alice", "Developer", age=30, city="New York", hobby="Reading")
```

Ключевое слово return

Ключевое слово `return` используется для возврата значения из функции в место её вызова. Оно завершает выполнение функции и возвращает указанное значение (или None, если значение не указано).

Особенности

- **Возврат значения:** `return` позволяет передать результат работы функции обратно к вызывающему коду.
- **Завершение функции:** После выполнения инструкции `return` функция завершает свою работу, даже если после неё есть другие строки кода.
- **Отсутствие значения:** Если `return` вызывается без указания значения, функция возвращает `None`.
- **Отсутствие return:** Даже если `return` отсутствует или не достигнут, функция выполняет код до конца и возвращает `None`.
- **Несколько return:** Функция может содержать несколько `return`. В этом случае выполнится первый достигнутый `return`.

Синтаксис

```
def function_name(parameters):
    # тело функции
    return value
```

Возврат значения

Пояснение

Функция `add` возвращает сумму чисел, которая сохраняется в переменной `result`.

Код

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


result = add(3, 5)
print(result)
```

Возврат одного из значений

Пояснение

Если условие выполняется, функция завершается после первого `return`, и код после него не исполняется.

Код

```
def check_positive(number):
    if number > 0:
        return "Положительное число"
    return "Отрицательное или ноль"

print(check_positive(10))
print(check_positive(-5))
```

Возврат None

Пояснение

Функция `say_hello` ничего не возвращает, поэтому её результат равен `None`.

Код

```
def say_hello():
    print("Hello, World!")


result = say_hello()
print(result)
```

Множественный возврат значений

Пояснение

`return` может возвращать несколько значений, которые упаковываются в кортеж.

Код

```
def calculate(a, b):

    return a + b, a - b


result = calculate(10, 5)

print(result)
```

Пустой return

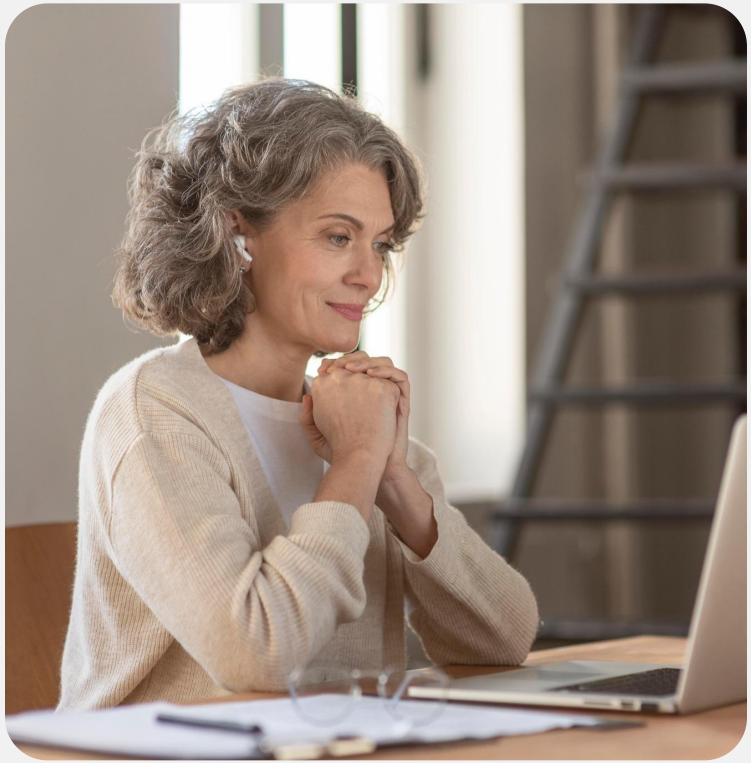
```
def calcula                                                    n):
    if n < 0:

        return  # Завершаем функцию без вычислений

    result = 1

    for i in range(1, n + 1):

        result *= i

    return result

num1 =                                                        -5
print(f"Факториал числа {num1}: {calculate_factorial(num1)}")

num2 =                                                         5
print(f"Факториал числа {num2}: {calculate_factorial(num2)}")
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

ЗАДАНИЕ

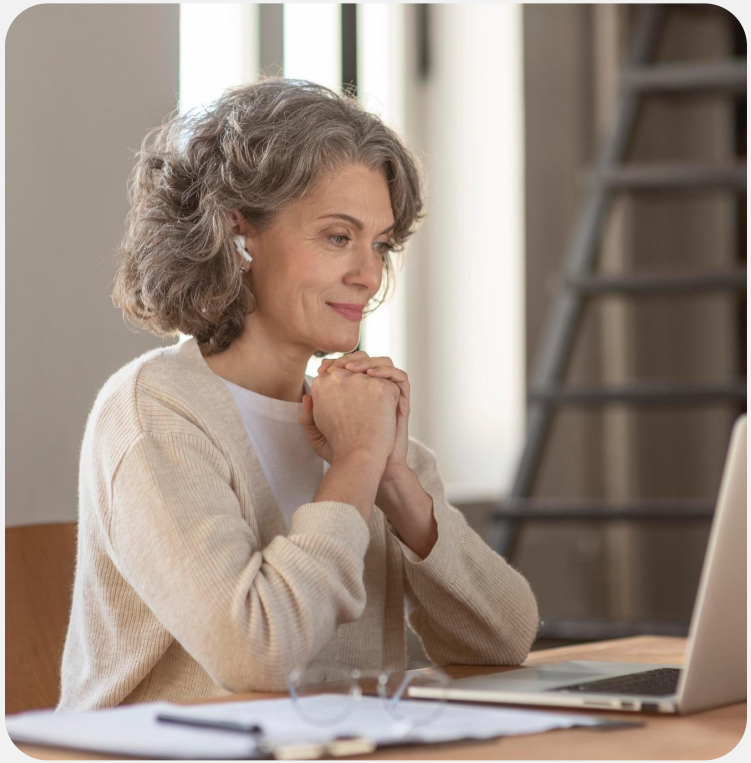


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
def example():

    pass



print(example())
```

a. None
b. Ошибка
c. pass
d. Ничего не будет выведено

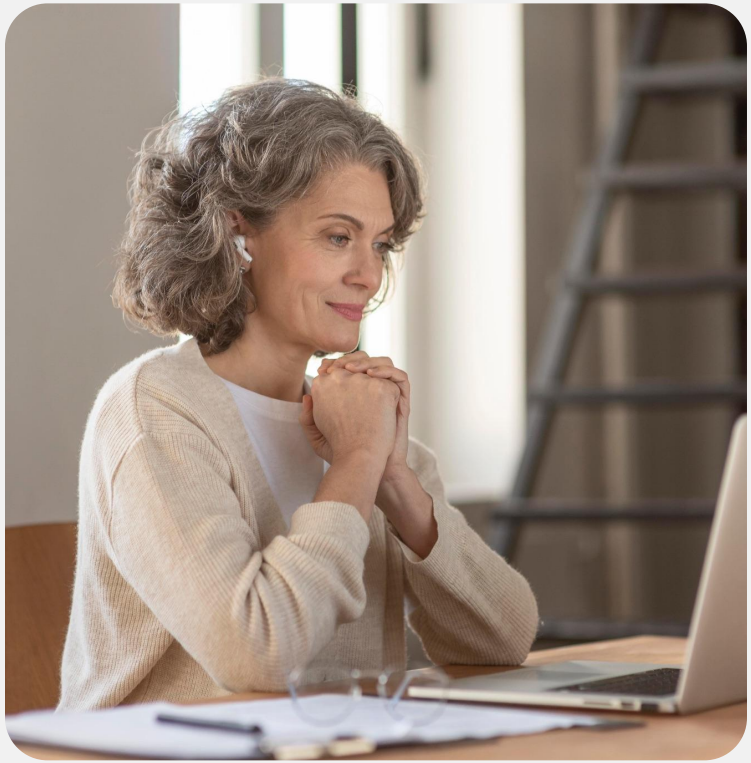


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
def example():

    pass



print(example())
```

a. None
b. Ошибка
c. `pass`
d. Ничего не будет выведено

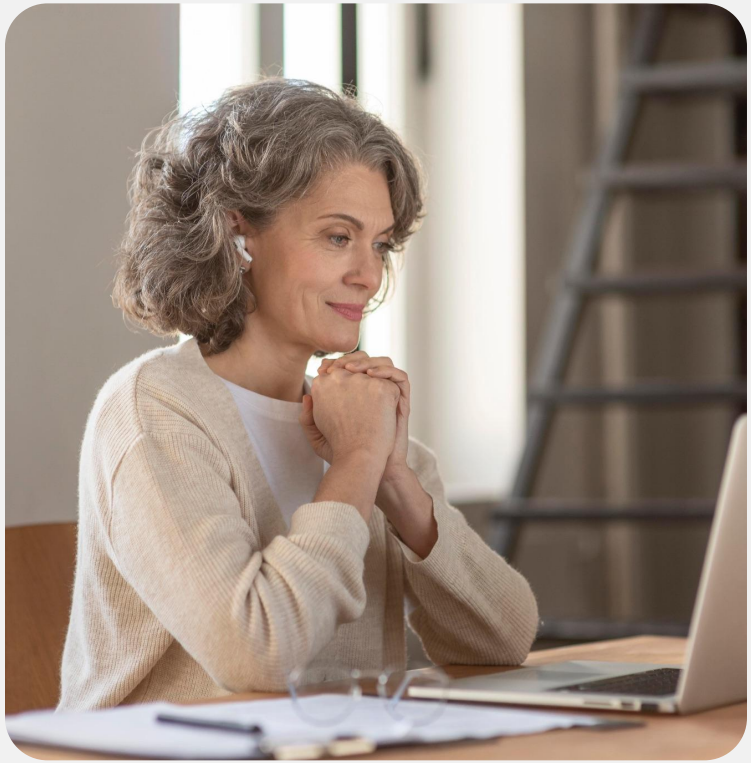


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
def func(a, b, c=10):

    return a + b + c



print(func(2, 3))
```

a. Ошибка
b. 15
c. 5
d. None

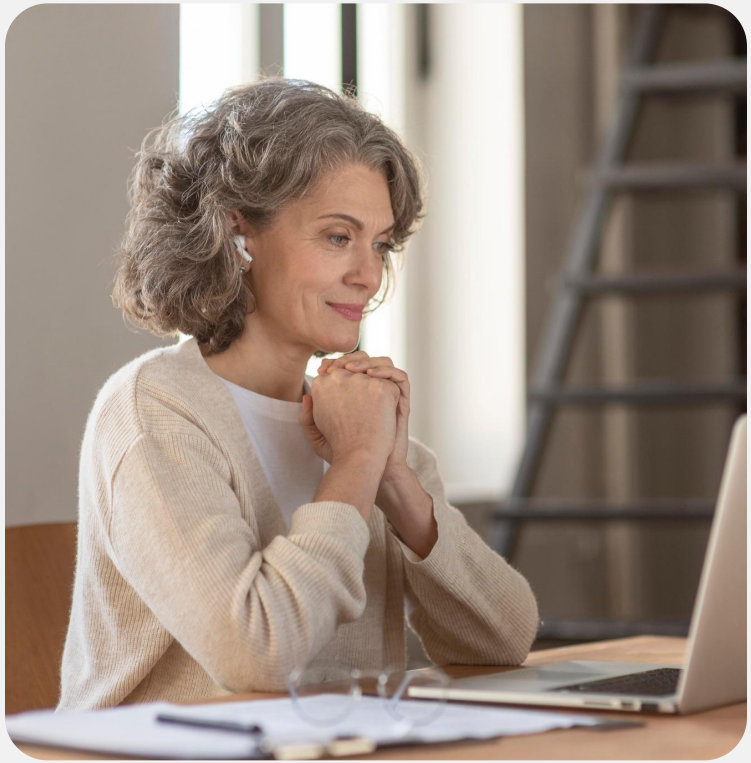


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
def func(a, b, c=10):

    return a + b + c



print(func(2, 3))
```

a. Ошибка
b. 15
c. 5
d. None

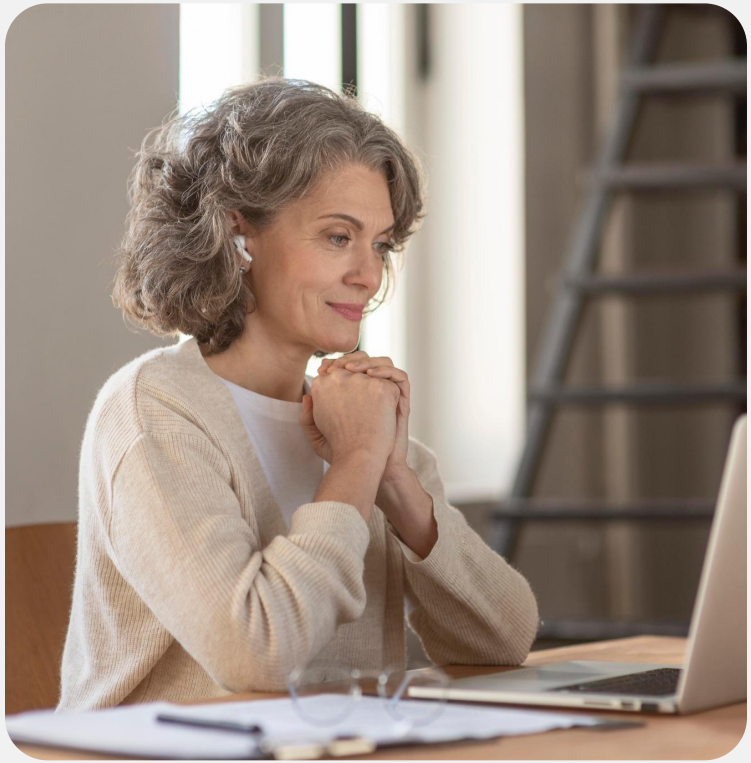


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
def check_number(n):
    if n > 0:
        return "Positive"
    return "Non-positive"
print(check_number(-1))
```

a. Ошибка из-за отсутствия `else`
b. `"Positive"`
c. `"Non-positive"`
d. `None`

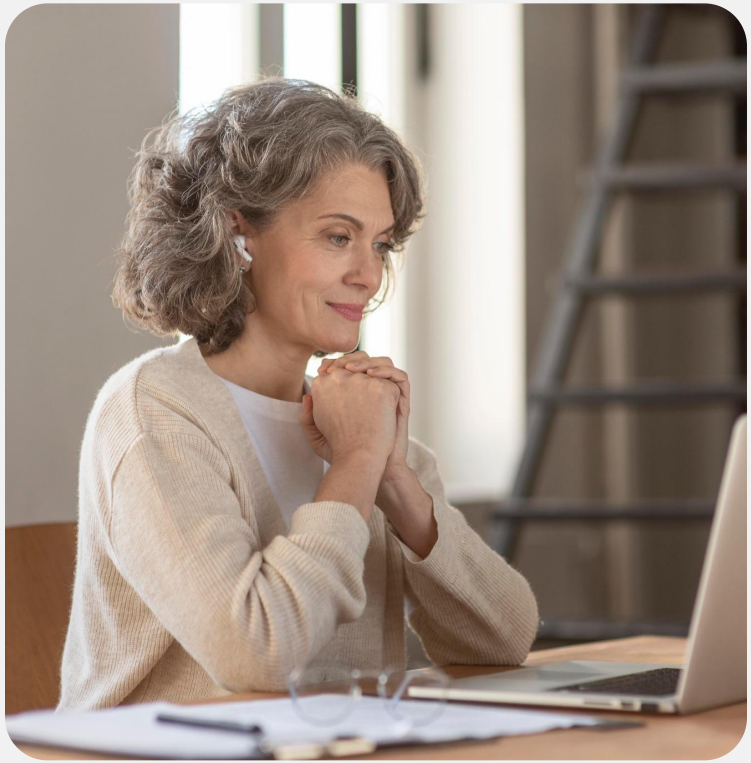


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
def check_number(n):
    if n > 0:
        return "Positive"
    return "Non-positive"
print(check_number(-1))
```

a. Ошибка из-за отсутствия `else`
b. `"Positive"`
c. `"Non-positive"`
d. `None`

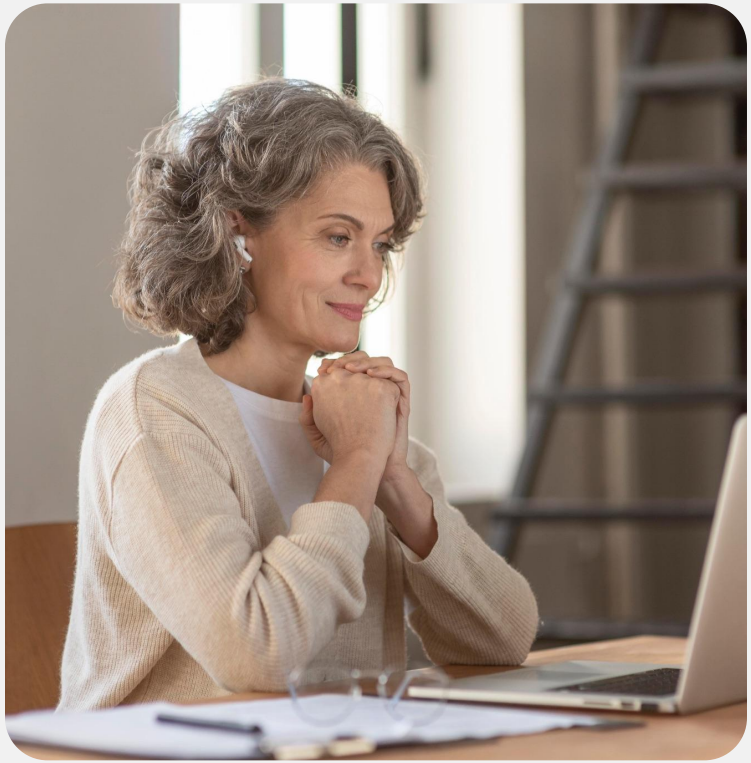


Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
def info(**kwargs):

    return kwargs


print(info(name="Alice", age=30))
```

a. Ошибка
b. `{"name": "Alice", "age": 30}`
c. `["name", "Alice", "age", 30]`
d. `("name", "Alice", "age", 30)`



Выберите правильный вариант ответа

Какой результат будет выведен при выполнении следующего кода?

```
def info(**kwargs):

    return kwargs


print(info(name="Alice", age=30))
```

a. Ошибка
b. {"name": "Alice", "age": 30}
c. ["name", "Alice", "age", 30]
d. ("name", "Alice", "age", 30)

IT CAREER HUB

ВОПРОСЫ

ICH IT CAREER HUB

Области видимости в Python

Полезно знать

ICH
IT CAREER HUB

Область видимости

Это контекст (границы), в котором переменные доступны для использования. Она определяет, где можно обращаться к данным и какие переменные доступны в разных частях программы.

Правило LEGB

- **Local** (Локальная область)
- **Enclosing** (Область окружающих функций, подробнее при изучении замыканий)
- **Global** (Глобальная область)
- **Built-in** (Встроенная область)

Local (Локальная область)

- **Область действия:** ограничена функцией, в которой переменная была объявлена.
- **Срок жизни:** существует только во время выполнения функции.
- **Определение:** локальная переменная создаётся внутри функции и доступна только в её пределах.

Пример

```python
def my_function():
    local_var = 10  # Локальная переменная
    print(f"Локальная переменная: {local_var}")

my_function()
# print(local_var)  # Ошибка: local_var недоступна за пределами функции
```

Global (Глобальная область)

- **Область действия:** доступна во всей программе, включая функции.
- **Срок жизни:** существует, пока выполняется программа.
- **Определение:** объявляется вне функций или других блоков и доступна во всей программе.

Пример

```python
global_var = 20  # Глобальная переменная


def show_global():
    print(f"Глобальная переменная: {global_var}")


show_global()
print(global_var)  # Глобальная переменная доступна в любом месте
```

Built-in (Встроенные объекты)

- Эта область содержит встроенные функции и объекты Python (например, `len()`, `print()`, `int()`).
- Они доступны в любом месте программы, если не переопределены.

Пример

```python
print(len("Hello"))  # Вызов встроенных функций len и print
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

Правило LEGB (поиск переменных в Python)

Python ищет переменные

- **Local** — в локальной области функции.
- **Enclosing** — в области окружающих функций (замыкания).
- **Global** — среди глобальных переменных.
- **Built-in** — среди встроенных объектов.

Примеры работы LEGB

Пояснение

Если переменная отсутствует во всех четырёх областях, **Python выбрасывает исключение** `NameError`.

Код

```
#   x   =   10      #   Глобальная   переменная
def function():

        #   x   =   5     #   Локальная   переменная
    print(x)

function()
```

Примеры работы LEGB

Пояснение

Если внутри функции объявляется переменная с тем же именем, что и глобальная, **локальная переменная перекрывает доступ к глобальной**.

Код

```
x = 10 # Глобальная переменная
def my_function():
    x = 5  # Локальная переменная с тем же именем
    print(f"Локальная переменная: {x}")

my_function()
print(f"Глобальная переменная: {x}")  # Глобальная переменная остаётся неизменной
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

Ключевое слово global

Полезно знать

ICH
IT CAREER HUB

global

Это ключевое слово в Python, которое позволяет изменять значение глобальной переменной внутри функции.

Примеры работы LEGB

Пояснение

variable_name — имя глобальной переменной, к которой требуется доступ.

Код

```
global variable_name
```

Пример без global

```
count = 0  # Глобальная переменная


def increment_counter():
    count = count + 1  # Ошибка, так как `count` внутри функции считается локальной
    print(count)


increment_counter()
```

Пример без global

Код вызовет **UnboundLocalError**, так как Python воспринимает `count` как локальную переменную, но мы пытаемся использовать её до присваивания.

Пример с global

```
count = 0  # Глобальная переменная


def increment_counter():
    global count  # Указываем, что работаем с глобальной переменной
    count += 1


increment_counter()
print(count)
increment_counter()
print(count)
```

Особенности использования global

- Изменяет глобальную переменную, а не создаёт новую локальную.
- Не требуется при **чтении** глобальной переменной внутри функции, но нужен при **изменении**.
- Чрезмерное использование `global` может привести к трудноотслеживаемым ошибкам, поэтому его следует применять с осторожностью.

Зачем передавать аргументы в функцию

В программировании рекомендуется передавать значения в функцию через параметры, а не использовать глобальные переменные внутри функции. Это связано с несколькими важными принципами и проблемами, которые могут возникнуть при работе с глобальными переменными.

Особенности использования global

- **Повышение читаемости и предсказуемости кода:** Когда функция получает значения через параметры, она становится независимой от внешних переменных.
- **Предотвращение неожиданных ошибок:** Использование глобальных переменных внутри функций может привести к неожиданным результатам.
- **Упрощение тестирования и переиспользования кода:** Функции, которые зависят от глобальных переменных, трудно тестировать.
- **Избегание конфликтов с именами переменных:** Глобальные переменные могут случайно перезаписаться в разных частях программы.

Пример с аргументами

```python
# Все значения передаются в функцию явно, что делает код понятным

def calculate_area(width, height):

    return width * height


result = calculate_area(5, 10)

print(result)
```

Пример с глобальными переменными

```
width = 5

height = 10


def calculate_area():

   # Непонятно, откуда берутся width и height, если смотреть только на функцию

   return width * height  # Использует глобальные переменные


result = calculate_area()

print(result)
```

Когда НЕ стоит использовать global

- **Для передачи данных между функциями** — лучше передавать параметры.
- **В больших программах** — сложно отслеживать, где изменяется переменная.
- **Для временных значений** — лучше использовать локальные переменные.

Когда global все же оправдан

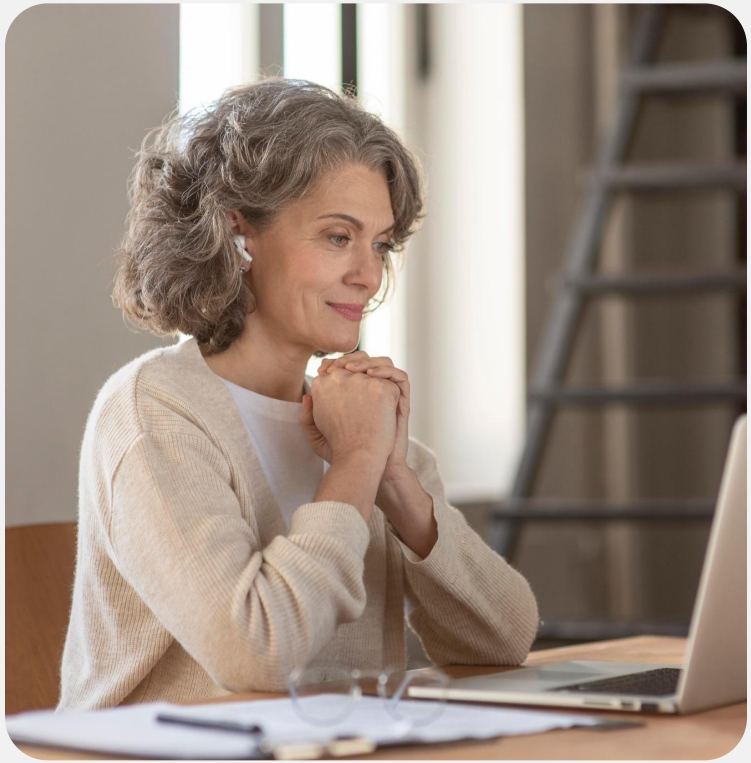
- **Для изменения переменных на уровне модуля.** Например, в настройках программы для хранения конфигураций.
- **В небольших скриптах**, где глобальные переменные помогают быстро решить задачу и масштабируемость не важна.

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

ПРАКТИЧЕСКИЕ ЗАДАНИЯ



Конвертер температуры

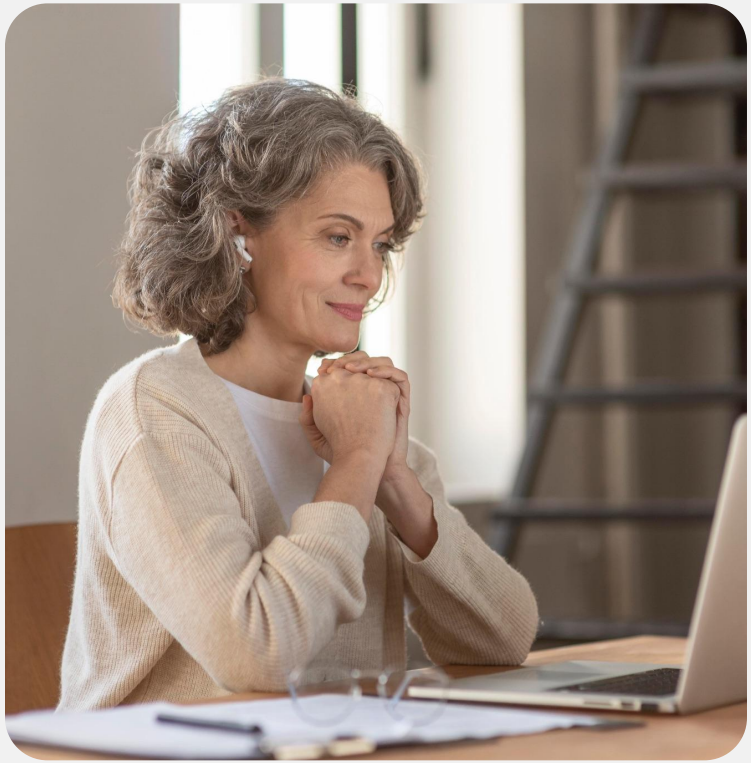
Напишите функцию, которая конвертирует температуру из градусов Цельсия в Фаренгейты и наоборот.

Формулы для конвертации температур:

- **Из градусов Цельсия в Фаренгейты:** F=C×95+32F = C \times \frac{9}{5} + 32
- **Из градусов Фаренгейта в Цельсия:** C=(F−32) ×59C = (F - 32) \times \frac{5}{9}

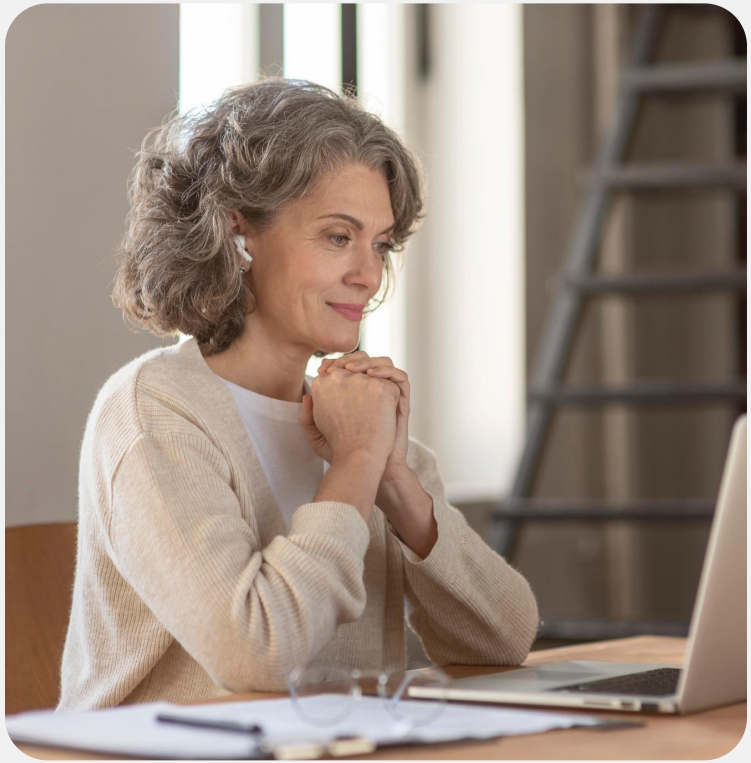
Данные:

```
temp = 100

scale = "C"
```



Решение

```
def convert_temperature(temp, scale):
    if scale.upper() == "C":
        return f"{temp}C = {temp * 9/5 + 32}F"
    elif scale.upper() == "F":
        return f"{temp}F = {(temp - 32) * 5/9}C"


temp = 100
scale = "C"
print(convert_temperature(temp, scale))
```



Фильтрация списка по длине

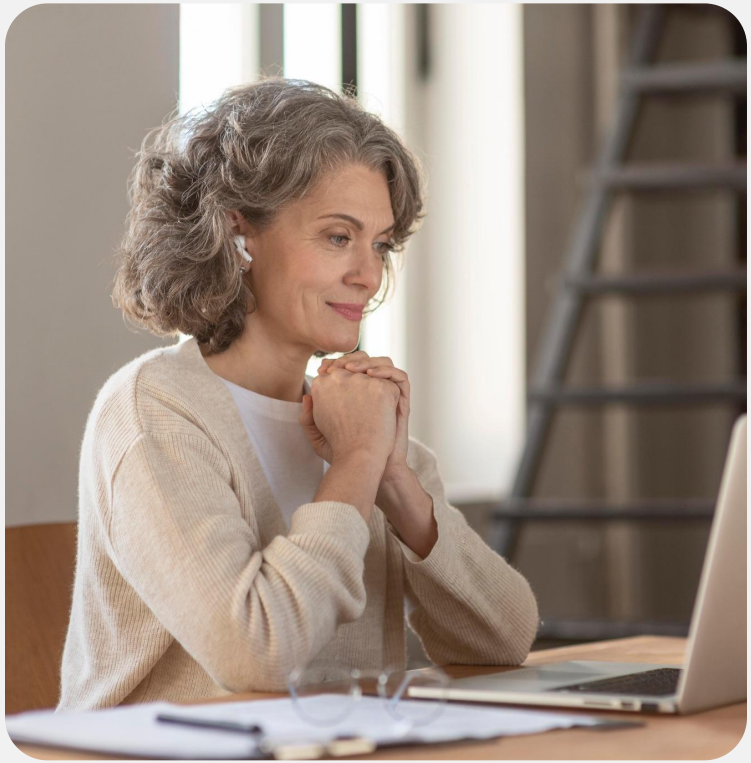
Напишите функцию, которая конвертирует температуру из градусов Цельсия в Фаренгейты и наоборот.

Формулы для конвертации температур:

- **Из градусов Цельсия в Фаренгейты:** F=C×95+32F = C \times \frac{9}{5} + 32
- **Из градусов Фаренгейта в Цельсия:** C=(F−32) ×59C = (F - 32) \times \frac{5}{9}

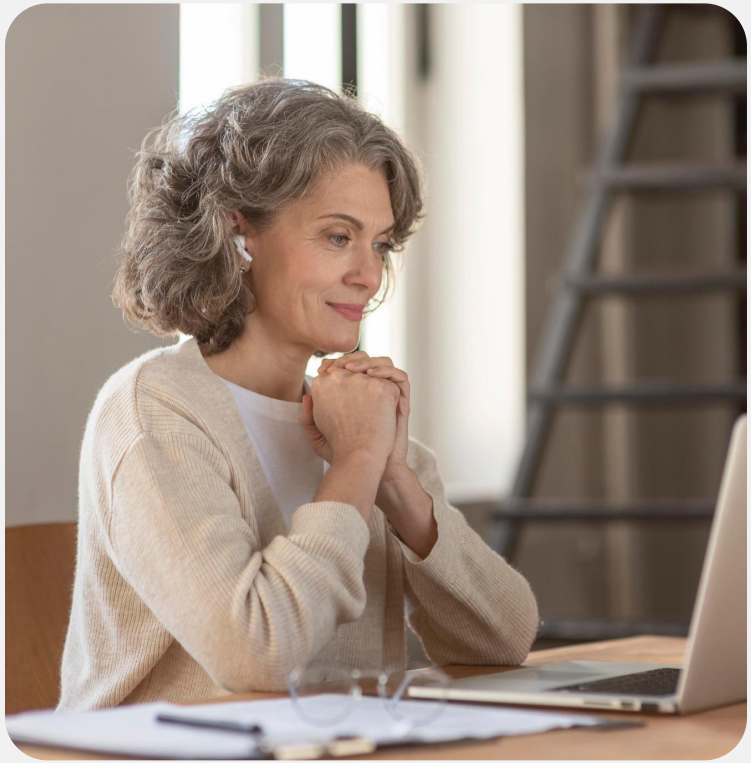
Данные:

```
temp = 100

scale = "C"
```



Решение

```
def filter_strings(min_len, *words):

    return [string for string in words if len(string) > min_len]


strings = ["apple", "banana", "cherry", "date", "fig"]

n = 5

print(filter_strings(n, *strings))
```



Фильтрация списка по длине

Проверка знака числа

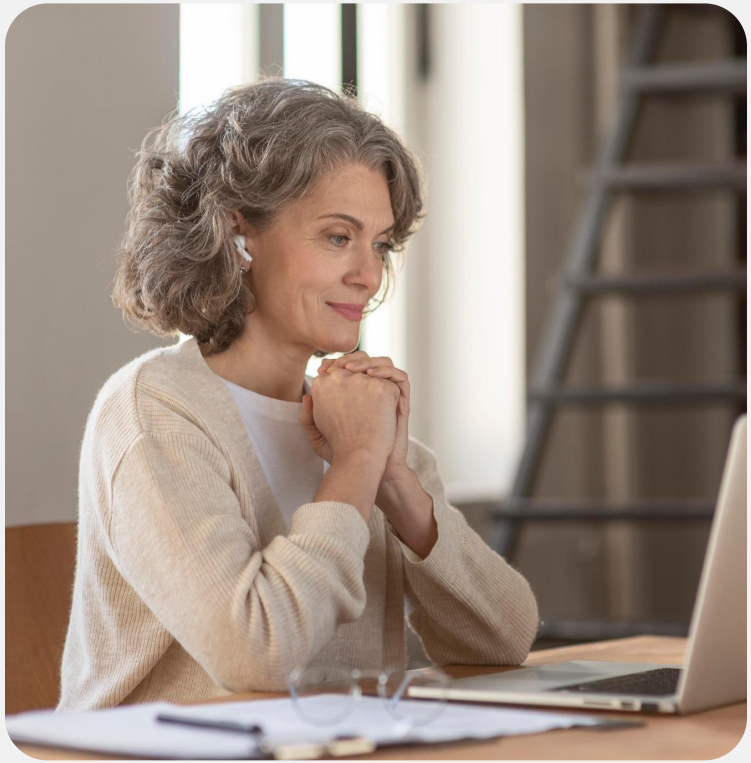
Напишите функцию, которая принимает число и возвращает, является ли оно **положительным, отрицательным или нулём**.

Данные:

```
num = -3
```

Пример вывода:

```
Число отрицательное
```



Решение

```
def check_number(num):
    if num > 0:
        return "Число положительное"
    elif num < 0:
        return "Число отрицательное"
    else:
        return "Число равно нулю"


num = -3
print(check_number(num))
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

ДОМАШНЕЕ ЗАДАНИЕ

Домашнее задание

Простое число

Напишите функцию, которая проверяет, является ли число n простым (делится только на 1 и само себя) и возвращает булевый результат.

Данные:

n = 17

Пример вывода:

Число 17 является простым

Домашнее задание

Фильтрация чисел по чётности

Напишите функцию, которая принимает filter_type ("even" или "odd") и произвольное количество чисел, возвращая только те, которые соответствуют фильтру.

Пример вызова:

```
print(filter_numbers("even", 1, 2, 3, 4, 5, 6))
print(filter_numbers("odd", 10, 15, 20, 25))
print(filter_numbers("prime", 2, 3, 5, 7))
```

Пример вывода:

```
[2, 4, 6]
[15, 25]
Некорректный фильтр
```

Домашнее задание

Объединение словарей

Напишите функцию, которая принимает любое количество словарей и объединяет их в один. Если ключи повторяются, используется значение из последнего словаря.

Данные:

```
dict1 = {"a": 1, "b": 2}
dict2 = {"b": 3, "c": 4}
dict3 = {"d": 5}
```

Пример вызова:

```
print(merge_dicts(dict1, dict2, dict3))
```

Заключение