ПМ3 Разработка модулей ПО.

РО 3.1 Понимать и применять принципы объектно-ориентированного и асинхронного программирования.

Тема 11. Функции.

Цель занятия:

Сформировать понимание назначения функций, научиться создавать и вызывать функции в Python, передавать аргументы, получать результаты и использовать функции для структурирования программ.

Учебные вопросы:

1. Понятие функции
2. Объявление и вызов функции.
3. Параметры и аргументы функции.
4. Возврат значений из функции.
5. Область видимости переменных.
6. Анонимные функции.
7. Документирование функций

1. Понятие функции.

Что такое функция?

Функция — это именованный блок кода, который выполняет определённую задачу и может быть использован многократно в разных частях программы.

Функция позволяет вынести повторяющиеся или логически связанные действия в отдельную структуру.

Что такое функция?

Функция сходить_в_магазин('магазин', список покупок)

1. Встать с дивана
2. Найти магазин на карте
3. Доехать до магазина
4. Купить товары по списку

Зафиксировать сумму затрат

сходить_в_магазин('Десяточка', [молоко, хлеб])

100 рублей

сходить_в_магазин('DNS', [мышь, клавиатура])

2000 рублей

Назначение функций.

Использование функций решает несколько важных задач:

- Повторное использование кода. Один и тот же код можно вызывать многократно без копирования.
- Структурирование программы. Программа становится более понятной и логичной.
- Упрощение отладки и сопровождения. Ошибки легче искать и исправлять в отдельных функциях.
- Повышение читаемости кода. Имена функций отражают смысл выполняемых действий.

Встроенные и пользовательские функции

Встроенные функции. Это функции, уже определённые в Python.

Примеры:

- **print()**
- **len()**
- **input()**
- **sum()**

Пользовательские функции. Функции, которые создаёт сам программист для решения конкретных задач.

Пример:

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

greet() # Hello!
```

Функция как «чёрный ящик».

Функцию можно рассматривать как чёрный ящик:

- на вход подаются данные (аргументы);
- внутри выполняются действия;
- на выходе возвращается результат.

Пример:

```
def square(x):
    return x * x

result = square(5) # 25
```

Связь функций с модульностью программы.

Программа, разбитая на функции:

- легче читается;
- проще тестируется;
- удобнее расширяется.

Функции являются основой модульного программирования.

Вывод:

- Функция — это основной инструмент структурирования программ.
- Использование функций делает код компактным, понятным и удобным для повторного использования.
- Понимание принципов работы функций является ключевым для дальнейшего изучения программирования.

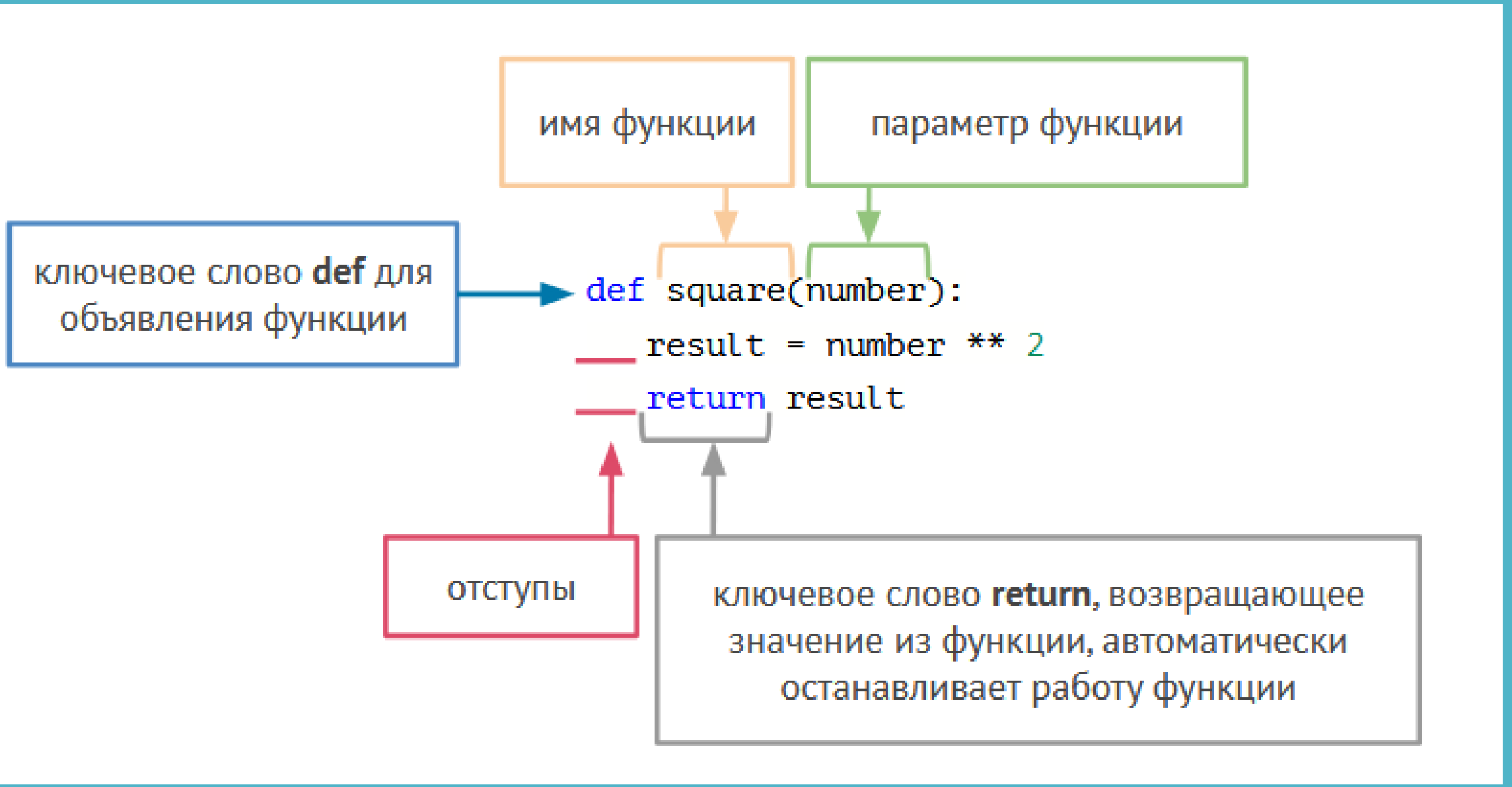
2. Объявление и вызов функции.

Для объявления (определения) функции в Python используется ключевое слово **def**. Общий синтаксис:

```
def имя_функции(параметры):
    тело_функции
```

где:

- def — ключевое слово объявления функции;
- имя_функции — имя функции (должно быть уникальным и отражать назначение);
- параметры — входные данные функции (могут отсутствовать);
- тело_функции — набор инструкций, выполняемых при вызове функции.


имя функции
параметр функции
ключевое слово def для объявления функции
def square(number):
result = number ** 2
return result
отступы
ключевое слово return, возвращающее значение из функции, автоматически останавливает работу функции

Пример:

```
1  def square(x):
2      return x * x
3
4  result = square(5) #
```

Здесь:

square – имя функции

x – параметр функции

square(5) – вызов функции с аргументом 5

Правила именования функций:

- имя функции должно начинаться с буквы или символа _;
- допускаются буквы, цифры и символ _;
- рекомендуется использовать snake_case;
- имя функции должно отражать выполняемое действие.

Примеры:

- print_message()
- read_input()
- calculate_sum()
- update_score()

Тело функции.

- тело функции обязательно должно быть с отступом (обычно 4 пробела);
- тело функции может содержать одну или несколько инструкций;
- если тело функции временно пустое, используется оператор **pass**.

```
def empty_function():
    pass
```

Вызов функции.

Функция вызывается по имени с указанием круглых скобок.

Пример:

```
def greet():
    print("Hello, world!")

greet() # Hello, world!
```

Последовательность выполнения:

- код внутри функции не выполняется, пока функция не будет вызвана;
- при вызове управление передаётся в тело функции;
- после завершения выполнения управление возвращается в место вызова.

Вывод:

- Функция объявляется с помощью ключевого слова **def**.
- Код функции выполняется только при её **вызове**.

3. Параметры и аргументы функции.

Функция может принимать входные данные — параметры.

```
def greet(name):
    print("Hello,", name)

greet("Anna")
```

Параметры — это переменные, указанные при объявлении функции.

Аргументы — это конкретные значения, которые передаются функции при её вызове.

```
def greet(name):    # name — параметр
    print("Hello,", name)

greet("Anna")       # "Anna" — аргумент
```

Позиционные аргументы

По умолчанию аргументы передаются в функцию по порядку, соответствующему порядку параметров.

```
def subtract(a, b):
    return a - b

result = subtract(10, 3)  # a=10, b=3
```

Порядок аргументов имеет значение.

Именованные (ключевые) аргументы

Аргументы можно передавать по имени параметра, явно указывая, какое значение к какому параметру относится.

```
def subtract(a, b):
    return a - b

result = subtract(b=3, a=10)
```

Преимущества:

- порядок аргументов не важен;
- код становится более читаемым.

Значения параметров по умолчанию

Параметрам можно задавать значения по умолчанию.

Если аргумент не передан при вызове, используется это значение.

```
def greet(name, greeting="Hello"):
    print(greeting, name)

greet("Anna")                   # Hello Anna
greet("Ivan", "Good day")       # Good day Ivan
```

Правило: параметры со значениями по умолчанию должны идти после обязательных параметров.

Нотация типов.

Нотация типов — это способ указать ожидаемые типы параметров и возвращаемого значения функции.

Она не влияет на выполнение программы, но:

- повышает читаемость кода;
- помогает обнаруживать ошибки;
- используется средами разработки и анализаторами кода.

Синтаксис нотации типов.

Тип параметра указывается после имени через двоеточие **:**.

Тип возвращаемого значения указывается после **->**.

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

Передача изменяемых объектов

Если аргументом является изменяемый объект (например, список), функция может изменить его содержимое.

```
def add_item(lst):
    lst.append(10)

numbers = [1, 2, 3]
add_item(numbers)
print(numbers)  # [1, 2, 3, 10]
```

Вывод:

- Параметры задаются при объявлении функции, аргументы — при её вызове.
- Аргументы могут быть позиционными или именованными.
- Значения по умолчанию делают функции более гибкими.
- Следует внимательно относиться к передаче изменяемых объектов.

4. Возврат значений из функции.

Назначение оператора return

Оператор **return** используется для:

- передачи результата работы функции обратно в место её вызова;
- завершения выполнения функции.

После выполнения **return** код внутри функции дальше не выполняется.

Возврат одного значения

Функция может возвращать одно значение любого типа.

```
def square(x: int) -> int:
    return x * x
    print(x * x) # это код никогда не вполнится

result = square(5)
print(result)
```

Использование возвращаемого значения

Результат функции можно:

- сохранить в переменную;
- использовать в выражении;
- передать в другую функцию.

```
def square(x):
    return x * x

total = square(3) + square(4)
print(total)
print(square(7))
```

Возврат нескольких значений

Функция может возвращать несколько значений одновременно.

Фактически возвращается кортеж.

```
def get_min_max(numbers: list) -> tuple:
    return min(numbers), max(numbers)

minimum, maximum = get_min_max([3, 7, 1, 9])
print(get_min_max([3, 7, 1, 9])) # (1, 9)
```

Возврат без значения

В Python функция всегда что-то возвращает.

Если в функции используется оператор **return** без выражения, функция возвращает специальное значение **None**, обозначающее *отсутствие значения*.

```
def show_message(text):
    print(text)
    return
```

Здесь **return** явно присутствует, но без значения. Функция вернет **None**.

Отсутствие оператора return

Оператор return может вообще не указваться, Python автоматически добавляет в конец функции:

return None

Поэтому следующий код эквивалентен предыдущему:

```
def show_message(text):
    print(text)
```

Досрочный выход из функции

return может использоваться для преждевременного завершения функции, например при проверке условий.

```
def divide(a, b):
    if b == 0:
        return None
    return a / b
```

Несколько операторов return

В функции может быть несколько операторов **return**, но выполняется только один — первый достигнутый.

```
def check_number(x):
    if x > 0:
        return "positive"
    elif x < 0:
        return "negative"
    return "zero"
```

Типичные ошибки:

- попытка использовать результат функции, которая ничего не возвращает;
- забытый **return** в конце функции;
- возврат значения другого типа, чем ожидается;
- размещение кода после **return**.

Вывод:

- **return** передаёт результат работы функции.
- Функция может возвращать одно или несколько значений.
- Отсутствие return означает возврат **None**.

5. Область видимости переменных.

Область видимости (scope) — это часть программы, где переменная доступна для использования.

В Python есть два основных типа областей видимости:

- Локальная — переменная доступна только внутри функции.
- Глобальная — переменная доступна во всей программе, включая функции (с некоторыми ограничениями).

Локальные переменные

- Объявляются внутри функции.
- Живут только во время выполнения функции.
- Недоступны снаружи функции.

```
def greet():
    message = "Hello"  # локальная переменная
    print(message)

greet()
# print(message)  # Ошибка: переменная не существует вне функции
```

Глобальные переменные

- Объявляются вне функций.
- Доступны в любой части программы, включая функции.

```
name = "Alice"  # глобальная переменная

def greet():
    print("Hello,", name)

greet()  # Hello, Alice
```

Время жизни переменной.

- Локальная переменная существует только во время вызова функции.
- Глобальная переменная существует в течение всей программы.

6. Анонимные функции. (Лямбда-функции (lambda))

Понятие лямбда-функции

- Лямбда-функция — это анонимная функция в Python, то есть функция без имени, которую можно объявить прямо в выражении.
- Используется для простых операций.
- Обычно применяется, когда функция нужна один раз.

Синтаксис

lambda параметры: выражение

- lambda — ключевое слово для создания функции;
- параметры — входные данные;
- выражение — возвращаемое значение (один результат).

Пример:

```
square = lambda x: x * x
print(square(5))  # 25
```

Здесь **square** — имя переменной, которая хранит лямбда-функцию.

Отличия от обычной функции

Параметр	Обычная функция	Лямбда-функция
Имя	Обязательно (def)	Не обязательно
Тело	Может содержать несколько строк	Только одно выражение
Возврат	Через return	Возвращает результат выражения автоматически
Использование	Для любых задач	Для простых, коротких операций

Примеры использования

Сортировка списка по ключу:

```
students = [("Ivan", 18), ("Anna", 20), ("Petr", 17)]
students.sort(key=lambda student: student[1])
print(students)
```

Использование в map и filter

```
numbers = [1, 2, 3, 4, 5]

# Умножение на 2
doubled = list(map(lambda x: x * 2, numbers))
print(doubled)

# Фильтрация чётных чисел
evens = list(filter(lambda x: x % 2 == 0, numbers))
print(evens)
```

Практическое применение:

- Кратковременные функции для map, filter, sorted;
- Логические операции и вычисления без объявления отдельной функции;
- Использование в качестве аргументов для других функций.

7. Документирование функций (docstring).

Документирование функции позволяет:

- объяснить, что делает функция;
- указать входные параметры и их типы;
- описать возвращаемое значение;
- облегчить использование функции другими разработчиками;
- использовать встроенные средства Python для просмотра описания.

Docstring — строка документации

- Docstring — это многострочная строка, помещённая сразу после объявления функции.
- Обрамляется тройными кавычками """ """ или ''' '''.
- Не влияет на выполнение функции.

```
def greet(name: str):
    """
    Функция выводит приветствие пользователю.

    Параметры:
    name (str): Имя пользователя

    Возвращаемое значение:
    None
    """
    print(f"Hello, {name}!")
```

Просмотр документации.

Можно использовать встроенную функцию **help()**:

```
help(greet)
```

```
Help on function greet in module __main__:

greet(name: str)
    Функция выводит приветствие пользователю.

    Параметры:
    name (str): Имя пользователя

    Возвращаемое значение:
    None
```

Контрольные вопросы:

- Что такое функция и зачем она используется?
- Чем отличается встроенная функция от пользовательской?
- Как объявить функцию в Python?
- В чём разница между параметрами и аргументами?
- Что делает оператор return?
- Как вернуть несколько значений из функции?
- Что такое локальная и глобальная переменная?
- Что произойдёт, если в функции нет оператора return?

Домашнее задание:

https://ru.hexlet.io/courses/js-asynchronous-programming

Материалы лекций:

https://github.com/ShViktor72/Education

Обратная связь:

colledge20education23@gmail.com