Тема 4. Функции и стандартные модули/библиотеки.

4.1 Функции в Python.

В python функция – это блок кода, который выполняет определенную задачу. Она позволяет группировать повторяющийся код и делает программу более организованной, читаемой и легко поддерживаемой. Функции в python могут иметь входные параметры(аргументы) и возвращать результаты.

Для создания функции в python используется ключевое слово «def», после которого следует имя функции и круглые скобки, в которых указываются аргументы функции (если они есть). Затем идет двоеточие, а после него блок кода функции, который выполняется при вызове функции.

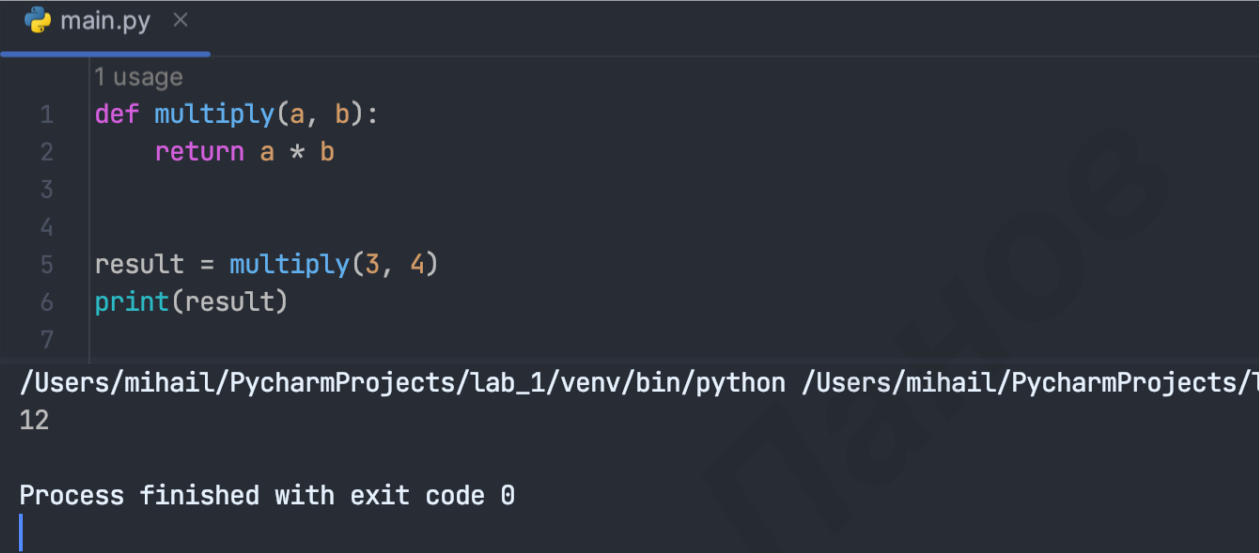
Функции могут принимать аргументы, которые передаются в скобках при их объявлении. Вот пример функции, которая принимает имя в качестве аргумента и выводит приветствие с использованием этого имени

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

В следующем примере мы определяем функцию с именем «multiply». Она принимает два аргумента – а и в, и возвращает их произведение с помощью оператора умножения \*.

Чтобы вызвать именную функцию, используйте ее имя, за которым следуют круглые скобки, содержащие значения аргументов. Вот пример вызова функции



В данном примере функция вызывается с аргументами 3 и 4, и возвращаемое ею значение сохраняется в переменную result. Затем это значение выводится на экран с помощью функции print

Конструкция return в python используется внутри функции для возврата значения из функции. Когда return выполняется, функция прекращает свое выполнение и возвращает указанное значение в вызывающую ее часть программы. Это позволяет использовать ее результаты работы функции в дальнейшем коде.

Вызов функций с именованными аргументами представлен ниже. В этом случае аргументы передаются по имени, что позволяет явно указать, какое значение относится к какому аргументу:



Вызов функции с переменным числом аргументов представлен ниже. Здесь функцию «add\_numbers» может принимать любое количество аргументов, переданных через запятую. Аргументы собираются в кортеж с помощью оператора “\*args”, и мы можем выполнять операции над ними.



Вызов функций с ключевыми аргументами представлен на рисунке ниже. В этом случае аргументы передаются по ключу, что позволяет указывать значения аргументов в любом порядке, не обязательно соблюдая исходный порядок объявления аргументов.



**4.2 Аргументы функции и параметры.**

Аргументы функции и параметры – это основные компоненты функции в python. Аргументы представляют значения, которые передаются функции при ее вызове, а параметры – это именованные переменные, которые принимают значения аргументов внутри функции. Когда функция вызывается в определении функции внутри круглых скобок после имени функции.

Функции могут иметь различные типы параметров:

**Позиционные параметры**: это параметры, значения которых передаются в том же порядке, в котором объявлены.

**Именованные параметры**: здесь значения передаются с указанием имени параметра. Это позволяет передавать аргументы в любом порядке и явно указывать, какое значение относится к какому параметру.

**Параметры со значением по умолчанию**: это параметры, которые имеют значение по умолчанию, которое будет использовано, если аргумент не будет передан при вызове функции

Подведем итоги: Разница между аргументами и параметрами в определении функции заключается в следующем:

Параметры – это именованные переменные, объявляемые в определении функции внутри круглых скобок после имени функции. Они служат для принятия и обработки значений, переданных функции при ее вызове. Параметры определяют, какие данные функция ожидает получить для своей работы.

Аргументы – это значения, передаваемые в функцию при ее вызове. Они передаются в соответствующие параметры функции, чтобы использовать внутри функции. Аргументы представляют фактические данные, которые передаются функции во время ее выполнения.

Обязательные аргументы – это аргументы, которые должны быть переданы функции при ее вызове. Если обязательный аргумент не будет передан, это может привести к ошибке.

Необязательные аргументы – это аргументы, для которых в определении функции устанавливается значение по умолчанию. Если необязательный аргумент не будет передан вызове функции, будет использовано значение по умолчанию. В python необязательные аргументы объявляются после обязательных аргументов и указываются с помощью имя\_параметра = значение\_по\_умолчанию.

**4.3 кортеж, как аргумент функции**

В python кортеж(tuple) представляет собой неизменяемую последовательность элементов, разделенных запятыми и заключенными в круглые скобки. Кортежи ведут себя подобно спискам, но не могут быть изменены после создания, что делает их неизменяемыми.

Кортежи поддерживают индексацию, что позволяет обратиться к элементам по их позиции. Индексация начинается с нуля.

Кортежи также поддерживают срезы (slicing), которые позволяют получить подмножество элементов из кортежа

Одной из особенностей кортежей является их неизменяемость. После создания кортежа нельзя изменить его элементы. Попытка присвоить новое значение элементу кортежа вызовет ошибку.

Однако если кортеж содержит изменяемые объекты, например, списки, то эти объекты могут быть изменены внутри кортежа, но сам кортеж останется неизменным.

Кортежи широко используются в python, особенно в ситуациях, когда требуется сохранить набор значений, которые не должны быть изменены. Кортежи могут быть возвращаемыми значениями функций, использоваться в качестве ключей словарей и использоваться в других контекстах, где неизменяемость данных является важным требованием.

В python кортежи могут быть переделаны в качестве аргументов функции точно так же, как и друге типы данных. Кортеж может быть передан в функцию как отдельный аргумент или вместе с другими аргументами.

Функции в python могут принимать неограниченное количество аргументов в том числе и кортежи. Для этого используется специальных синтаксис оператором \*.

Таким образом, кортежи могут быть использованы в качестве аргументов функций в python, их элементы могут быть обработаны внутри функции или переданы в другие функции для дальнейшей обработки.

**4.4 Анонимные функции lambda.**

В python есть возможность создавать анонимные функции с помощью ключевого слова lambda. Анонимные функции также известны как lambda-функции, представляют собой компактный способ определения функции без использования стандартного синтаксиса определения функций с помощью def

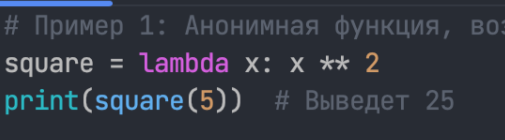
Общий синтаксис для создания анонимных функций выглядит следующим образом:

**lambda arguments: expression**

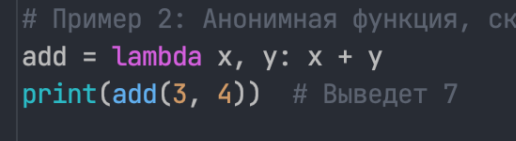
Arguments – список аргументов, которые принимает анонимная функция.

Expression – выражение, которое выполняется внутри анонимной функции и возвращает результат

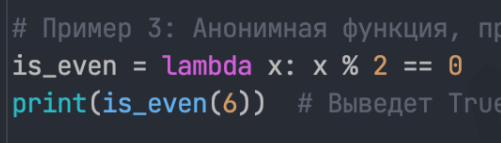
В первом примере создается анонимная функция square, которая принимает аргумент х и возвращает квадрат этого числа. Мы присваиваем эту функцию переменной square и вызываем ее, передавая значение 5



Во втором примере создается анонимная функция add, которая принимает два аргумента х и у и возвращает их сумму. Мы присваиваем эту функцию переменной add и вызываем ее, передавая значение 3 и 4.



В третьем примере создается анонимная функция is\_even, которая принимает аргумент х и проверяет, является и число четным. Она возвращает True, если число четной и False, если число нечетное. Мы присваиваем эту функцию переменной is\_even и вызываем ее, передавая значение 6



Анонимные функции особенно полезны, когда требуется определить небольшие функции на лету или передать их в качестве аргументов другим функциям, таким как map(), filter(), reduce() и другим. Они обычно используются в ситуациях, где необходимо выполнить некоторые простые операции без необходимости определения именованной функции с помощью def.

**Рекурсивные функции.**

Рекурсивные функции в программировании – это функции, которые вызывают сами себя в своем теле. Это особый подход к решению задач, когда задачи разбивается на более простые подзадачи, и каждая подзадача решается вызовом той же функции. Рекурсия является мощным инструментом для решения определенных типов задач и часто используется в алгоритмах и структурах данных

Основные характеристики рекурсивных функций:

**Базовый случай:** каждая рекурсивная функция должна иметь базовый случай, когда вызов функции прекращается и она возвращает результат напрямую, без дополнительных рекурсивных вызовов. Это предотвращает бесконечное повторение вызовов и обеспечивает завершение рекурсии

**Рекурсивный случай:** когда условие базового случая не выполняется, функция вызывает себя снова с измененными аргументами. Это позволяет сократить сложную задачу до простых подзадач, которые решаются рекурсивными вызовами.

Рекурсивные функции могут быть мощным инструментом для решения определенных задач, но их следует использовать с осторожностью, так как неправильная реализация может привести к переполнению стека вызовов (stack overflow) или бесконечной рекурсии. Важно правильно определить базовый случай и убедиться, что рекурсивные вызовы приводят к сокращению задачи до базового случая

**4.6 Глобальные переменные**

Глобальные переменные в python – это переменные, которые определены за пределами всех функций и доступны из любого места в программе. Они могут быть использованы и изменены внутри любой функции или блоке кода

Объявление глобальной переменной происходит путем присваивания значения переменной вне функции или блоков кода.

Внутри функций можно использовать глобальные переменные без необходимости объявления их как аргументы функции или локальные переменные. Однако, если вы попытаетесь изменить значение глобальной переменной внутри функции, не указав ее как глобальную, будет создана новая локальная версия с тем же именем и глобальная переменная останется неизменной. Для того, чтобы явно указать, что вы хотите изменить значение глобальной переменной, внутри функции используйте ключевое слово global.

Хотя использование глобальных переменных может быть удобным в некоторых случаях, их следует использовать с осторожностью, так как они могут усложнить отслеживание и понимание программы. Модификация глобальных переменных внутри функций может привести к неожиданному поведению и усложнить отладку программы. Поэтому рекомендуется использовать глобальные переменные только в случае, когда они действительно необходимы и оправданы. В большинстве случаев предпочтительнее использовать локальные переменные и передавать значение через аргументы функций.

**4.7 Документирование функции**

В python существует соглашение о документировании функций с помощью строки документации, которая помещается внутри тройных кавычек. Эта строка документации представляет описание функции, ее параметры, возвращаемое значение строки и другую полезную информацию

Строка документации может содержать следующие элементы:

**Описание функции:** общее описание того, что делает функция и ее назначение.

**Аргументы(параметры):** описание каждого аргумента функции, указывающее тип данных и его роль.

Возвращаемое значение: описание значения, которое возвращает функция.

Примеры использования: примеры кода, которые демонстрируют, как использовать функцию.

**Другая информация:** любая другая полезная информация, связанная с функцией. Документируя функции с помощью строки документации, вы облегчите понимание и использование функции другим разработчикам, а также помогаете себе в будущем, когда вам нужно будет вернуться к функции и вспомнить ее основные детали.

Доступ к строке документации функции осуществляется с помощью атрибута \_\_doc\_\_. В данном примере это выведет содержимое строки документации функции.

Обратите внимание, что документация является хорошей практикой программирования, но она не является обязательной. Однако, ее использование считается хорошим стилем кодирования и помогает создавать более чистый и поддерживаемый код.

4.8 **присвоение функций переменной**

В python функции можно присваивать переменным так же, как и любые другие значения. Это означает, что вы можете создать функцию и присвоить ее переменной, а затем использовать эту переменную для вызова функции

Присвоение функций переменной полезно, когда вам нужно передать функцию как аргумент в другую функцию или сохранить функцию для последующего вызова

Присваивание функции переменной является мощным инструментом в Python, который позволяет гибко работать с функциями и использовать их в различных сценариях программирования

**4.9 Модули**

В python модуль – это файл, содержащий код (функции классы, переменные) и предназначенный для организации и управления кодом в более крупных программных проектах. Модели позволяют разделить код на логически связанные блоки и обеспечить повторное использование кода

Преимущества использования модулей в Python:

**Организация использования кода:** модели позволяют логически разделить функциональность программы на отдельные файлы, что облегчает его понимание, обслуживание, и развитие

**Повторное использование кода:** Модули позволяют создавать функции, классы и переменные, которые можно повторно использовать в различных частях программы или в других проектах

**Разделение обязанностей:** модули позволяют разделить код между несколькими разработчиками или командами, так что каждый модуль может быть независимо разрабатываемым и поддерживаемым

Для использования модуля в Python вы должны выполнить следующие шаги:

* Создайте файл с расширением .ру в котором будет содержаться код модуля.
* Определите функции, классы, и переменные внутри файла модуля
* В другом файле или интерактивной оболочке Python используйте оператор import для импорта модуля и доступа к его содержимому

Обратите внимание, что файл с расширением .ру в котором будет содержаться код модуля должен находиться в той же папке что и основной файл

Кроме простого импорта с помощью import, существуют другие способы импорта модулей, например: from module import item. Этот способ позволяет импортировать только определенные элементы из модуля, а не весь модуль целиком

Модули являются важной частью организации кода в python, позволяя создавать масштабируемые и структурированные программные проекты. Они предоставляют возможность разделить код на логические блоки, повторно использовать код и упростить совместную работу между разработчиками

**4.10 Глобальное использование функций модуля**

Чтобы сделать функции модуля доступными для глобального использования в других модулях или в основной программе, вы можете использовать ключевое слово from в операторе импорта или импортировать модуль целиком

**Импортирование модуля целиком**. В этом случае вы импортируете весь модуль. Вы можете обращаться к функциям модуля, используя синтаксис module\_name.function\_name() или module \_name.variable\_name

**Импортирование конкретных функций из модуля.** В этом случае вы импортируете только функции, которые вам нужны из модуля. Теперь вы можете использовать эти функции без необходимости указывать имя модуля перед их именем.

**Импортирование всего модуля с псевдонимом.** В этом случае вы импортируете модуль с псевдонимом. Теперь вы можете использовать функции модуля, префиксируя их имена псевдонимом.

Это лишь несколько способов использования функций модуля глобально. Важно помнить, что глобальное использование функций модуля может упростить код, но также может создать конфликты и неоднозначности, если имена перекрываются с другими именами в вашей программе. Поэтому важно выбирать имена функций и модулей таким образом, чтобы избежать возможных конфликтов и создать читаемый и понятный код

В python вы можете импортировать один или несколько модулей в одной строке, разделяя их запятыми. Например:

* Import my\_module;
* Import module1; module 2;
* Import module 1 as m1, module2 as m2, module3 as m3
* Import math

Import random, datetime, os

Import numpy as np, pandas as pd

Также в python существует ключевое слово import \*, оно используется для импорта всех функций, классов, и переменных из модуля в текущее пространство имен. Это позволяет обращаться к импортированным элементам напрямую, без необходимости указывать имя модуля перед ним

Однако, использование import \* не рекомендуется в большинстве случаев по нескольким причинам:

**Неясность имен**: использование import\* может привести к неоднозначности имен в вашем коде. Если модуль, из которого вы импортируете, содержит функции или переменные с именами, которые уже используются в вашей программе, это может привести к конфликтам и трудностям в понимании откуда именно эти имена происходят.

**Загромождение пространства имен:** Импортирование всех элементов из модуля с помощью import\* может привести к загромождению вашего пространства имен. Если модуль содержит много функций и переменных, это может сделать ваш код менее читабельным и усложнить отслеживание происхождения и использования определенных имен

Непредсказуемость: Использование import\* делает ваш код менее предсказуемым и может затруднить отслеживание и анализ зависимостей между модулями. При чтении кода другим разработчикам может быть сложно определить, откуда именно происходят импортированные элементы, и какие модули необходимы для правильной работы программы.

Использование import\* не рекомендуется поскольку это может привести к неясности имен, загромождению пространства имен и непредсказуемости кода

**4.11 Точка входа в Python**

Точка входа в python – это основная часть программы, с которой начинается ее выполнение. Она определяет, какой код будет запущен при запуске программы.

В python точка входа обычно определена в виде вызова функции main(). Это позволяет разделит логику программы на отдельные функции и выполнить основную логику в функции main()

В python переменная \_\_name\_\_ представляет специальное внутреннее имя, которое предоставляется интерпретатором python для каждого модуля. Значение переменной \_\_name\_\_ зависит от контекста, в котором используется.

Когда модуль выполняется как точка входа (т.е. запускается напрямую), значение переменной \_\_name\_\_ будет равно \_\_main\_\_. Это позволит определить является ли текущий модуль точкой входа в программу. Например, вы можете использовать условие if \_\_name\_\_ == ‘\_\_main\_\_’: для выполнения определенного кода только при запуске модуля напрямую, а не при его импортировании в другой модуль

Когда модуль импортируется в другой модуль, значение переменной \_\_name\_\_ будет равно имени самого модуля. Это позволяет отличить запуск модуля от его импорта и, например, вызвать определенные функции или классы из импортированного модуля. В этом примере модуль main\_module импортирует модуль module и вызывает функцию some\_function() из него только при запуске main\_module напрямую. Если main\_module импортируется в другой модель, код в блоке if \_\_name\_\_ == ‘\_\_main\_\_’: не будет выполняться.

Таким образом, переменная \_\_name\_\_ представляет информацию о том, в каком контексте выполняется модуль, и позволяет контролировать выполнение определенного кода в зависимости от этого контекста.

**4.12 Рекурсивный импорт**

Рекурсивный импорт – это ситуация, когда два или более модуля взаимно импортируют друг друга непосредственно или через цепочку импортов. Такой сценарий может возникнуть, если в модуле А импортируется модуль В, а в модуле В импортируется модуль А.

Рекурсивный импорт может привести к проблемам и ошибкам, таким как “importError: cannot import name” или “RuntimeError: maximum recursion depth exceeded”. Это связано с тем, что python пытается разрешить импорт внутри циклической зависимости, но не может это сделать, так как один модуль еще не полностью загружен при попытке импорта другого модуля.

Для избежания рекурсивного импорта можно пересмотреть структуру кода и выделить общую функциональность в третий модуль или переорганизовать зависимости между модулями. В некоторых случаях также может быть полезно использовать отложенный импорт (lazy import) или импортировать модули внутри функции, чтобы избежать рекурсивных проблем.

При попытке импортировать модуль А или модуль В произойдет ошибка рекурсивного импорта. Решение проблемы рекурсивного импорта требует пересмотра архитектуры программы и избегания прямы или косвенных циклических зависимостей между модулями

**4.13 Встроенные модули Python**

Python поставляется со множеством встроенных модулей, которые предоставляют различные функциональные возможности. Вот несколько примеров встроенных модулей Python:

1. ‘math’: представляет математические функции и константы, такие как тригонометрические функции, логарифмы, возведение в степень и другие операции
2. ‘random’: позволяет генерировать случайные числа, выбирать случайные элементы из списков и выполнять другие операции связанные со случайными значениями
3. ‘datetime’: представляет классы и функции для работы с датами, вычислять разницу между датами, формировать даты и времена и многое другое.
4. ‘os’: представляет функции для взаимодействия с операционной системой. Этот модуль позволяет получать информацию о файловой системе, выполнять операции с файлами и папками работать с переменными окружения и выполнять другие операции связанные с операционной системой
5. ‘sys’: предоставляет функции и переменные, связанные с интерпретатором Python. Этот модуль позволяет получать информацию о версии python, аргументах командной строки, путях поиска модулей и других системных параметрах
6. ‘json’: предоставляет функции для работы с данным в формате JSON. Этот модуль позволяет сериализовывать объекты python в формат JSON и десериализовывать JSON-данные в объекты python
7. ‘re’: предоставляет поддерddжку регулярных выражений в python. Этот модуль позволяет выполнять поиск, сопоставление и манипуляции текстом с использованием регулярных выражений
8. ‘csv’: предоставляет функции для работы с данными CSV (command separated values). Этот модуль позволяет читать данные из csv-файлов, записывать данные в сым-файлы и выполнять другие операции, связанные с csv- данными.

Это лишь некоторые примеры встроенных модулей python. Полный встроенных библиотек python можно найти в официальной документации python.аа