**ВОПРОСЫ К ЭКЗАМЕНУ ПО ДИСЦИПЛИНЕ**

**«АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ В ЯЗЫКЕ PYTHON»**

**(ПЕРВЫЙ СЕМЕСТР)**

**ВОПРОС ПЕРВЫЙ (ТЕОРЕТИЧЕСКИЙ), 20 БАЛЛОВ**

**1. Специфика типизации в языках программирования (различные аспекты типизации). Реализация типизации в Python.**

Суть типизации в том, как языки программирования распознают переменные в кодификации. Вид типизации определяет работу с данными – придется ли разработчику вручную указывать тип данных или машина это сделает сама.

Языки делятся на типизированные и нетипизированные. К примеру в языке Forth, который является нетипизированным, все данные записываются цепочками бит произвольной длины, что в свою очередь позволяет работать с любыми типами данных и делает код более эффективным.

Типизация может быть статической или динамической – от данного параметра будет зависеть, когда компилятор вычисляет тип переменных и ищет ошибки. Это может происходить:

- на этапе компиляции – сначала код переводиться на машинный язык, а затем программа готовится к запуску (статическая типизация);

- этап исполнения — код выполняется с момента запуска программы (динамическая типизация).

Преимущества статической типизации:

* Тип переменной проверяется только при компиляции. Это дает разработчикам следующее преимущество: нет необходимости постоянно контролировать, не пытается ли программа, например, умножить число на строку и другие ошибочные действия.
* Скорость исполнения. Динамически типизированные языки программирования медленнее, чем статически типизированные.
* Зачастую ошибки удаётся обнаружить уже на стадии компиляции.
* При использовании IDE (интегрированная среда поддержки) программисты могут эффективно и быстро разрабатывать программы, отсеивая данные неподходящего типа.

Основные плюсы динамической типизации:

* Легко создавать универсальные коллекции. Такая задача стоит нечасто, но именно динамическая типизация поможет ее решить наиболее эффективно.
* Удобно составлять алгоритмы обобщения. К примеру, сортировка массива. При этом он будет выполняться не только с численными переменными, но и со строками и вещественными данными.
* Просто освоить. Такие языки программирования считаются самыми подходящими для новичков.

Слабая и строгая типизация.

Слабая типизация позволяет смешивать разные типы данных, т.к. язык сам производит преобразование данных. При строгой типизации разработчик сам прописывает все операции с типами данных.

Плюсы строгой типизации:

* Основательность — машина выдаст ошибку в случае неправильных действий.
* Оперативность — при строгой типизации все преобразования прописываются максимально прозрачно. Именно поэтому программист знает все медленные и наиболее затратные участки кода.
* Знание всех тонкостей программы. Разработчик пишет ее вручную и понимает тонкости каждой операции.
* Конкретность — при такой форме типизации программист прекрасно ориентируется в том, какие данные он преобразует, а также насколько вероятны неточности.

Преимущества слабой типизации:

* Удобство работы со смешанными типами данных. Например, целочисленные и вещественные числа.
* Акцент на выполнении конкретной задачи.
* Краткость кода.

Явная и неявная типизация.

Данные параметры отвечают за то, необходимо ли объявлять типы данных в коде. При явной типизации (С++) объявляем тип данных, при неявной (Python) это сделает компилятор.

Плюсы явной типизации:

* Каждая функция имеет сигнатуру (int add(int, int)), что даёт возможность быстро понять, что конкретно она делает.
* Разработчик изначально пишет, какие значения соответствуют переменной данного типа. Нет необходимости запоминать их.

Плюсы неявной типизации:

* Краткий код. Например, def add(x, y) вместо int add( int x, int y).
* Устойчивость к правкам. В случае если в функции временная переменная была того же типа, что и входной аргумент, то при правках последнего необходимо изменить и тип временной переменной.

Бестиповая типизация (низкоуровневые языки программирования).  
Данные записываются в цепочку битов и не подвергаются типизации.  
У данного направления типизации есть несомненные плюсы. Среди них:

* У разработчика есть возможность писать коды на предельно низком уровне. Компилятор не выдаст ошибку и не запустит проверку типов данных. Есть возможность осуществлять любые действия над переменными различного типа.
* Код показывает высокую практическую эффективность.
* Код прозрачен. Если язык программирования известен, то любой код понятен программисту.

Минусы:

* Сложность. Зачастую необходимо представить комплексные значения, например, списки или структуры. Это достаточно трудоемкий процесс.
* Нет проверок. Даже самые нецелесообразные действия считаются нормальными. К примеру, вычитание указателя на массив из символа. Ошибки, которые порождает данное действие могут быть незаметными.
* Низкая абстрактность. Работа над типами данных разной сложности не будет отличаться, что чревато определенными сложностями для разработчика.

Язык Python является типизированным, причем обладает неявной, строгой (операции между разными типами происходят по определенным правилам), динамической типизацией.

**2. Присвоение по ссылке и по значению. Специфика создания объектов и присвоения в Python, особенности Python в связи с распространенностью использования неизменяемых типов.**

Передача по значению означает, что когда мы передаем переменную в функцию, фактически передается копия значения этой переменной. Это означает, что изменения, которые происходят с переменной внутри функции, не влияют на исходное значение.

Передача по ссылке, наоборот, означает, что передается ссылка на переменную, а не ее значение. Изменения, которые происходят с переменной внутри функции, влияют на исходное значение.

Неизменяемые объекты передаются по значению. Это значит, что при изменении значения переменной будет создан новый объект. К этому типу относятся:

- числовые данные (int, float, complex)

- символьные строки (str)

- кортежи (tuple)

При инициализации переменной неизменяемого типа создается объект (например, целое число), этот объект имеет некоторый идентификатор: оператор = связывает переменную a и объект посредством ссылки. При этом вы не можете изменить сам объект, т.е. когда вы присвоите переменной новое значение, интерпретатор создаст новый объект (если до этого этот объект был создан, то переменная просто получит ссылку), а первоначальный объект удалится из памяти сборщиком мусора, если ссылок на него больше нет.

Изменяемые объекты передаются по ссылке. Это значит, что при изменении значения переменной объект будет изменен. К этому типу относятся:

списки (list)

множества (set)

словари (dict)

>>> a = 10

>>> id(a)

10914656

Стоит отметить, что для объектов малого значения в языке Python, в целях повышения эффективности кода используется кеширование, что в свою очередь означает, что независимо связанные ссылки будут ссылаться на один и тот же объект.

Поскольку язык Python строго типизирован при работе с неизменяемыми типами данных имеются определенные ограничения при работе с неизменяемыми типами данных. Операции с разными типами данных невозможны, за исключением случаев автоматического преобразования типов данных по определенным правилам в результате работы интерпритатора.

**3. Списки в Python. Обращение к элементам списка и создание срезов.**

Список это изменяемая, итерируемая, упорядоченная коллекция в Python.

Список представляет собой последовательность элементов, пронумерованных от 0, как символы в строке. Для создания списка используется конструктор list.

Создание списков:

 явно указав все элементы списка в тексте (элементы списка перечисляются в квадратных скобках через запятую):

 с помощью функции list():

 поэлементное заполнение списка, используя метод append(), который добавляет в конец списка указанное значение:

Операции над списками:

 Конкатенация списков (добавление одного списка в конец другого).

 Повторение списков (умножение списка на число), повторяет список указанное количеств раз.

Проверка на вхождение (in, not in). Оператор in проверяет наличие значения в списке, оператор not in — отсутствие значения. Возвращают значения True или False.

Для обращения к элементам списка надо использовать индексы, которые представляют номер элемента в списке. Индексы начинаются с нуля. То есть первый элемент будет иметь индекс 0, второй элемент - индекс 1 и так далее. Для обращения к элементам с конца можно использовать отрицательные индексы, начиная с -1. То есть у последнего элемента будет индекс -1, у предпоследнего - -2 и так далее.

 Извлечение среза. Со списками, также как и со строками, можно делать срезы. А именно:

o A[i : j] — срез из j – i элементов A[i], A[i + 1], …, A[j – 1].

o A[i : j : –1] — срез из i – j элементов A[i], A[i-1], …, A[j + 1] (то есть меняется порядок элементов).

o A[i : j : k] — срез с шагом k: A[i], A[i + k], A[i + 2\*k], … . Если значение k<0, то элементы идут в противоположном порядке.

Каждое из чисел i или j может отсутствовать, что означает «начало строки» или «конец строки».

Операции со списками в Python:

 x in A — проверить, содержится ли элемент в списке. Возвращает True или False.

 x not in A — то же самое, что not(x in A).

 min(A) — наименьший элемент списка.

 max(A) — наибольший элемент списка.

 A.index(x) — индекс первого вхождения элемента x в список, при его отсутствии генерирует исключение ValueError.

 A.count(x) — количество вхождений элемента x в список.

 A.sort() — сортировка списка (меняет сам список, ничего не возвращает).

 sum(A) — возвращает сумму элементов в списке.

 A.append(x) — добавить элемент x в конец списка.

 A.extend(L) — добавить все элементы списка L в конец списка A.

Синтаксис генераторов списков в Python:

[выражение for переменная in список],

где переменная — идентификатор некоторой переменной, список — список значений, который принимает данная переменная (как правило, полученный при помощи функции range), выражение — некоторое выражение, которым будут заполнены элементы списка, как правило, зависящее от использованной в генераторе переменной.

Метод строки split() возвращает список строк, разрезав исходную строку на части по пробелам по умолчанию: A = input().split(). Если при запуске этой программы ввести строку 1 2 3, то список A будет равен ['1 ', '2 ', '3 '].

У метода split есть необязательный параметр, который определяет, какая строка будет использоваться в качестве разделителя между элементами списка. Например, метод split(';') вернет список, полученный разрезанием исходной строки по символам '; '.

Метод строки join(lst) выводит список при помощи однострочной команды. У этого метода один параметр: список строк. В результате получается строка, полученная соединением элементов списка (которые переданы в качестве параметра) в одну строку, при этом между элементами списка вставляется разделитель, равный той строке, к которой применяется метод.

**4. Кортежи в Python. Отличия кортежей от списков. Распаковка и частичная распаковка кортежей.**

Кортеж – это неизменяемая, индексируемая, упорядоченная коллекция в Python.

Выводы:

1. Кортеж определяется так же, как список, за исключением того, что набор элементов заключается в круглые скобки, а не в квадратные.

2. Элементы кортежа заданы в определённом порядке, как и в списке. Элементы кортежа индексируются с нуля, как и элементы списка, таким образом, первый элемент не пустого кортежа — это всегда a\_tuple[0].

3. Отрицательные значения индекса отсчитываются от конца кортежа, как и в списке.

Последний элемент имеет индекс –1.

4. Создание среза кортежа («slicing») аналогично созданию среза списка. Когда создаётся срез списка, получается новый список; когда создаётся срез кортежа, получается новый кортеж.

Особенности кортежа:

 Нельзя добавить элементы к кортежу. Кортежи не имеют методов append() или extend().

 Нельзя удалять элементы из кортежа. Кортежи не имеют методов remove() или pop().

 Можно искать элементы в кортежи, поскольку это не изменяет кортеж.

 Можно использовать оператор in, чтобы проверить существует ли элемент в кортеже.

Встроенная функция tuple() принимает список и возвращает кортеж из всех его элементов, функция list() принимает кортеж и возвращает список.

Кортежи в логическом контексте можно использовать в операторе if.

 В логическом контексте пустой кортеж является ложью.

 Любой кортеж, состоящий, по крайней мере, из одного элемента, — истина.

 Любой кортеж, состоящий, по крайней мере, из одного элемента, — истина. Значения элементов не важны.

 Чтобы создать кортеж из одного элемента, необходимо после него поставить запятую. Без запятой Python предполагает, что вы просто добавили еще одну пару скобок, что не делает ничего плохого, но и не создает кортеж.

**5. Множества в Python. Основные способы создания, получения и изменения значений. Обход множеств.**

Множество – это изменяемая, неиндексируемая, неупорядоченная коллекция в Python, с которой можно сравнивать другие элементы, чтобы определить, принадлежат ли они этому множеству. Множество может состоять из различных элементов, порядок элементов во множестве не определен.

Элементами множества может быть любой неизменяемый тип данных: числа, строки, кортежи. Изменяемые типы данных не могут быть элементами множества, в частности, нельзя сделать элементом множества список (но можно сделать кортеж) или другое множество. Каждый элемент может входить во множество только один раз, порядок задания элементов не важен.

Задание множеств:

 множество задается перечислением всех его элементов в фигурных скобках.

 с помощью функции set()

Операции над множествами:

 Чтобы проверить, принадлежит ли значение множеству, используйте оператор in. Он работает так же, как и для списков.

 Метод union() (объединение ( | )) возвращает новое множество, содержащее все элементы каждого из множеств.

 Метод intersection() (пересечение ( & )) возвращает новое множество, содержащее все элементы, которые есть и в первом множестве, и во втором.

 Метод difference() (разность ( – )) возвращает новое множество, содержащее все элементы, которые есть во множестве a\_set, но которых нет во множестве b\_set.

 Метод symmetric\_difference() (симметрическая разность ( ^ )) возвращает новое множество, которое содержит только уникальные элементы обоих множеств.

Примеры применения операций:

 copy() — создает копию множества.

 add() — добавляет во множество.

 remove() — удаляет из множества. Если элемент не найден, то возбуждается исключение KeyError.

 discard() — удаляет из множества, если он присутствует.

 pop() — удаляет произвольный элемент из множества и возвращает его. Если элементов нет, то возбуждается исключение KeyError.

 clear() — удаляет все элементы из множества.

Элементы множества можно обойти в цикле for:

С множествами в Python можно выполнять обычные для математики операции над множествами:

 A |= B или A.update(B) — добавляет во множество A все элементы из множества B.

 A &= B или A.intersection\_update(B) — оставляет во множестве A только те элементы, которые есть во множестве B.

 A –= B или A.difference\_update(B) — удаляет из множества A все элементы, входящие во множество B.

 A ^= B или A.symmetric\_difference\_update(B) — записывает в A симметрическую разность множеств A и B.

 A <= B или A.issubset(B) — возвращает True, если A является подмножеством B.

 A >= B или A.issuperset(B) — возвращает True, если A является надмножеством B.

 A < B — эквивалентно A <= B and A != B.

 A > B — эквивалентно A >= B and A != B.

При использовании формы A.update(B) и аналогичных, параметр B может также являться списком, строкой и вообще любым итерируемым объектом. Так, запись A.update([1, 2, 1]) корректна (во множество будут добавлены элементы 1 и 2), а запись A |= [1, 2, 1] — породит исключение.

**6. Словари в Python. Итерирование по словарям, преобразование между словарями и списками в Python.**

Словарь – это изменяемая, неиндексируемая, неупорядоченная коллекция элементов в Python, состоящая из пар ключ-значение. Словари создаются с помощью фигурных скобок {} и могут быть изменяемыми.

Способы создания словарей:

 использование фигурных скобок {}:

 использование функции dict():

 использование функции zip():

 использование генератора словаря:

 использование метода fromkeys():

Операции над словарями:

 доступ к элементам по ключу:

 проверка наличия ключа в словаре:

 del: удаляет из словаря элемент с заданным ключом. Если элемент с таким ключом отсутствует, то возникает ошибка (возбуждается исключение KeyError).

Основные методы работы со словарями:

 keys(): возвращает список всех ключей в словаре.

 values(): возвращает список всех значений в словаре.

 items(): возвращает список кортежей (пар ключ-значение) в словаре.

 get(): возвращает значение, связанное с указанным ключом. Если ключ не существует, возвращает указанное значение по умолчанию.

 pop(): удаляет элемент из словаря по указанному ключу и возвращает его значение.

Если ключ не найден, возвращает указанное значение по умолчанию.

 update(): обновляет словарь, добавляя пары ключ-значение из другого словаря или итерируемого объекта.

 clear(): удаляет все элементы из словаря.

Перебор элементов словаря. Перебор всех элементов словаря выполняется с помощью цикла for. Если в заголовке цикла указать просто имя словаря, то переменная цикла на каждой итерации будет равна ключу очередного элемента словаря.

Если использовать в заголовке цикла метод values(), то переменная цикла на каждой итерации будет равна значению очередного элемента словаря.

Если в заголовке цикла использовать метод keys(), то переменная цикла на каждой итерации будет равна ключу очередного элемента словаря.

Если использовать в заголовке цикла метод items(), то переменные цикла (k и v) на каждой итерации будут равны ключи и соответствующему значению очередного элемента словаря.

**7. Циклы в Python, работа и устройство цикла for, типичное применение range и enumerate в цикле for.**

Цикл for пробегается по набору значений, помещает каждое значение в переменную, и затем в цикле мы можем с этой переменной производить различные действия.

Синтаксис цикла вида for:

for переменная in объект:

инструкции

else:

инструкции\_else

Блок else может отсутствовать.

При выполнении цикла Python последовательно получает все значения из набора и передает их переменную. Когда все значения из набора будут перебраны, цикл завершает свою работу.

Для цикла for применимы функции range, которая генерирует числовую последовательность, которая становится целочисленный итератором и enumerate, которая позволяет “пересчитать» итерируемый объект. Её можно использовать не только для обработки списков, но и для работы с другими типами данных — со строка‐ми, кортежами, словарями.

Функция range возвращает целочисленный итератор. Способы обращения к функции range:

- range(stop) — c одним аргументом (stop) итератор представляет последовательность целых чисел от 0 до (stop – 1).

- range (start, stop) — с двумя аргументами (start, stop) итератор представляет последовательность целых чисел от start до (stop – 1).

- range (start, stop, step) — с тремя аргументами итератор представляет последовательность целых чисел от start до (stop – 1) с шагом step.

Функция enumerate обычно используется в циклах for...in, чтобы получить последовательность кортежей (index, item), где значения индексов отсчитывается от 0 или от значения start.

Синтаксис функции enumerate:

- enumerate(i) генерируется последовательность кортежей (index, item), где значения индексов отсчитывается от 0.

- enumerate(i, start) генерируется последовательность кортежей (index, item), где значения индексов отсчитывается от значения start

**8. Циклы в Python, работа и устройство цикла while**.

Цикл while («пока») позволяет выполнять одну и ту же последовательность действий, пока проверяемое условие истинно. Условие записывается до тела цикла и проверяется до выполнения тела цикла. Как правило, цикл while используется, когда невозможно заранее определить точно количество шагов.

Синтаксис цикла while:

while условие:

блок инструкций

else:

инструкции\_else

При выполнении цикла while сначала проверяется условие. Если оно ложно, то выполнение цикла прекращается и управление передается на следующую инструкцию после тела цикла while. Если условие истинно, то выполняется инструкция, после чего условие проверяется снова и снова выполняется инструкция. Так продолжается до тех пор, пока условие будет истинно. Как только условие станет ложно, работа цикла завершится и управление передастся следующей инструкции после цикла.

Использование инструкции break в теле цикла приводит к немедленному прекращению цикла, при этом не выполняется ветка else. Следующей будет выполняться инструкция, следующая сразу за циклом.

Использование инструкции continue в теле цикла приводит к тому, что все оставшиеся инструкции тела цикла пропускаются, происходит переход на строку заголовка и проверка условия цикла. Далее все выполняется как обычно.

**9. Обход списка и поиск элементов в списке. Ключевые операции, проводящие к изменению списка и порождающие измененные списки, копирование списков.**

Обход списка происходит с помощью цикла for. Функция enumerate() позволяет удобно обходить список и оперировать с текущим индексом элемента.

Проверка на вхождение (in, not in). Оператор in проверяет наличие значения в списке, оператор not in — отсутствие значения. Возвращают значения True или False.

Для обращения к элементам списка надо использовать индексы, которые представляют номер элемента в списке. Индексы начинаются с нуля. То есть первый элемент будет иметь индекс 0, второй элемент - индекс 1 и так далее. Для обращения к элементам с конца можно использовать отрицательные индексы, начиная с -1. То есть у последнего элемента будет индекс -1, у предпоследнего - -2 и так далее.

Также возможен поиск элемента по значению с помощью метода index(), который возвращает индекс элемента (если значения нет в списке, то возбуждается исключение ValueError).

Основные операции, изменяющие список:

 A.append(x) — добавить элемент x в конец списка.

 A.extend(L) — добавить все элементы списка L в конец списка A

insert (<Индекс>, <Объект>) добавляет один объект в указанную позицию.

Конкатенация списков.

Умножение списка на число (в итоге в полученный список входит n исходных списков, n число, на которое умножаем список)

С помощью среза

рор([<Индекс>]) - удаляет элемент, расположенный по указанному индексу, и возвращает его.

* Если индекс не указан, то удаляет и возвращает последний элемент списка.
* Если элемента с указанным индексом нет или список пустой, возбуждается исключение *IndexError*.

**Инструкция** del может удалять из списка как единичные элементы, так и элементы, получаемые при помощи среза

Копирование списков:

Список можно скопировать с помощью конструктора списков list(). В таком случае происходит поверхностное копирование.

Также можно сделать глубокую копию с помощью метода deepcopy из библиотеки copy.

**10. Организация условий в языке Python, тернарный оператор.**

Для организации условий в языке Python используются такие логические конструкции как логическое сложение (ИЛИ), логическое умножение (И), логическое отрицание (НЕ - not), а так же следующие операторы сравнения:

< - меньше,

> - больше,

<= - меньше или равно,

>= - больше или равно,

== - равенство,

!= - неравенство.

Для создания условного (тернарного) оператора используется конструкция, использующая операнды if, elif, else. Тернарный операнд возвращает значение второго или третьего своего операнда, в зависимости от значения логического выражения, заданного первым операндом. Общий вид тернарного оператора в Python: true\_result if {инструкция\_1} else false\_result

**11. Объявление и вызов функции в Python. Параметры функции со значением по умолчанию и комментирование функции. Получение информации о функции. Способы передачи параметров при вызове функции.**

Функции в Python — это блоки кода, которые могут принимать аргументы, выполнять определенные действия и возвращать результаты. Они позволяют группировать код для выполнения конкретной задачи и повторно использовать его в программе. Функции делают код более модульным, упрощают его чтение и обеспечивают повторное использование кода.

Функция в Python имеет следующую структуру:

def function\_name(argument1, argument2, ...):

# Код функции

return result

- def — это ключевое слово, которое говорит о начале определения функции.

- function\_name — это имя функции, которое должно быть допустимым идентификатором.

- argument1, argument2, ... — это аргументы функции, это значения, которые можно передать в функцию для ее выполнения.

- return — это ключевое слово для возврата результата.

Функция вызывается с помощью конструкции:

имя\_функции([значения\_параметров]).

Также мы можем сохранить ссылку на функцию в другую переменную и вызывать её через данную переменную.

## Документирование функций

echo.\_\_doc\_\_

После объявления функции мы можем написать свой многострочный текст. Доступ к нему можно получить:

* по команде ?
* с помощью функции help()
* с помощью свойства \_\_doc\_\_

Среди разработчиков на Python популярно несколько стандартов документирования функции. Использование стандартов позволяет автоматически генерировать документацию с помощью специальных инструментов.

**reST**

Возможно наиболее распространенный стандарт. Формат используется инструментом **Sphinx** (<https://www.sphinx-doc.org/en/master/>) для генерации документации. Этот формат по умолчанию используется IDE PyCharm.

Пример:

"""

This is a reST style.

:param param1: this is a first param

:param param2: this is a second param

:returns: this is a description of what is returned

:raises keyError: raises an exception

"""

**Google**

Компания Google поддерживает собственный формат комментариев функций. Он тоже может использоваться инструментом Sphinx (необходимо использование плагина Napoleon).

Пример:

"""

This is an example of Google style.

Args:

param1: This is the first param.

param2: This is a second param.

Returns:

This is a description of what is returned.

Raises:

KeyError: Raises an exception.

"""

**Numpydoc**

Библиотека NumPy рекомендует использовать собственный стиль описания функций (базируется на стиле Google и может использоватьс в Sphinx).

Пример:

"""

My numpydoc description of a kind

of very exhautive numpydoc format docstring.

Parameters

----------

first : array\_like

the 1st param name `first`

second :

the 2nd param

third : {'value', 'other'}, optional

the 3rd param, by default 'value'

Returns

-------

string

a value in a string

Raises

------

KeyError

when a key error

OtherError

when an other error

"""

**Epytext**

Стиль поддерживаемый инструментом для генерации документации Epydoc (<http://epydoc.sourceforge.net>) называется форматом Epytext format.

Пример:

"""

This is a javadoc style.

@param param1: this is a first param

@param param2: this is a second param

@return: this is a description of what is returned

@raise keyError: raises an exception

"""

Объекты функций поддерживают множество **атрибутов**. Обратиться к атрибутам функции можно, указав атрибут после названия функции через точку. Например, через атрибут \_\_nаme\_\_ можно получить название функции в виде строки, через атрибут \_\_doc\_\_ – строку документирования и т. д.

['\_\_annotations\_\_',

'\_\_builtins\_\_',

'\_\_call\_\_',

'\_\_class\_\_',

'\_\_closure\_\_',

'\_\_code\_\_',

'\_\_defaults\_\_',

'\_\_delattr\_\_',

'\_\_dict\_\_',

'\_\_dir\_\_',

'\_\_doc\_\_',

'\_\_eq\_\_',

'\_\_format\_\_',

'\_\_ge\_\_',

'\_\_get\_\_',

'\_\_getattribute\_\_',

'\_\_getstate\_\_',

'\_\_globals\_\_',

'\_\_gt\_\_',

'\_\_hash\_\_',

'\_\_init\_\_',

'\_\_init\_subclass\_\_',

'\_\_kwdefaults\_\_',

'\_\_le\_\_',

'\_\_lt\_\_',

'\_\_module\_\_',

'\_\_name\_\_',

'\_\_ne\_\_',

'\_\_new\_\_',

'\_\_qualname\_\_',

'\_\_reduce\_\_',

'\_\_reduce\_ex\_\_',

'\_\_repr\_\_',

'\_\_setattr\_\_',

'\_\_sizeof\_\_',

'\_\_str\_\_',

'\_\_subclasshook\_\_']

Способы задания функции в Python:

- без аргументов:

- с одним аргументом:

- с несколькими аргументами:

- с аргументами по умолчанию:

- передача аргументов с использованием их имени:

- возвращение нескольких значений:

**12. Передача переменного количества параметров (именованных и не именованных) в функции Python. Вызов функции с позиционными параметрами, находящимися в списке, и именованными параметрами, находящимися в словаре**.

\*args используется для передачи неопределенного числа неименованных аргументов. Если поставить звездочку перед именем, это имя будет представлять собой кортеж из всех переданных аргументов функции. С помощью

\*\*kwargs работает так же, как и \*args, но вместо кортежа используется словарь. Это позволяет функции принимать любое количество именованных аргументов.

Операторы распаковки. Если мы хотим передавать функции несколько именованных или неименованных параметров, то используем их в определении аргументов функции. Если хотим передать параметры из списка (кортежа) в функцию с позиционными аргументами, то применяем их при вызове функций.

**13. Функции стандартной библиотеки для работы с контейнерами.**

В Python стандартная библиотека предоставляет множество функций для работы с контейнерами, такими как списки, кортежи, множества и словари. Рассмотрим наиболее полезные из них.

### Для всех контейнеров:

len() – возвращает количество элементов в контейнере.

sorted() – возвращает отсортированную копию контейнера.

reversed() – возвращает итератор, который генерирует элементы в обратном порядке.

all() – возвращает True, если все элементы контейнера истинны.

any() – возвращает True, если хотя бы один элемент контейнера истинный.

sum() – возвращает сумму всех элементов контейнера (для числовых типов).

min() **и** max() – возвращают минимальное и максимальное значение в контейнере соответственно.

enumerate() – создает итератор, который возвращает кортежи (индекс, элемент) для каждого элемента в контейнере.

### Для списков:

append() – добавляет элемент в конец списка.

extend() – расширяет список элементами из другого контейнера (например, списка).

insert() – вставляет элемент на заданную позицию.

pop() – удаляет элемент по индексу и возвращает его.

remove() – удаляет первый встреченный элемент по значению.

index() – возвращает индекс первого вхождения элемента.

count() – возвращает количество вхождений элемента в список.

reverse() – инвертирует элементы списка на месте.

copy() – возвращает поверхностную копию списка.

### Для кортежей:

index() – возвращает индекс первого вхождения элемента.

count() – возвращает количество вхождений элемента в кортеж.

### Для множеств:

add() – добавляет элемент в множество.

remove() – удаляет элемент, вызывает ошибку, если элемента нет.

discard() – удаляет элемент, не вызывает ошибку, если элемента нет.

pop() – удаляет и возвращает произвольный элемент множества.

clear() – удаляет все элементы из множества.

union() – возвращает объединение двух множеств.

intersection() – возвращает пересечение двух множеств.

difference() – возвращает разницу двух множеств.

### Для словарей:

get() – возвращает значение по ключу, если ключ не найден, возвращает None (или заданное значение).

keys() – возвращает представление всех ключей в словаре.

values() – возвращает представление всех значений в словаре.

items() – возвращает представление всех пар (ключ, значение) в словаре.

update() – обновляет словарь элементами из другого словаря или итерируемого объекта.

pop() – удаляет и возвращает элемент по ключу.

popitem() – удаляет и возвращает пару (ключ, значение).

clear() – удаляет все элементы из словар

Эти функции и методы значительно упрощают работу с контейнерами в Python и обеспечивают гибкость при манипуляциях с данными.

**14. Выполнение основных операций с парой множеств в Python.**

С множествами в Python можно выполнять обычные для математики операции над множествами:

 A |= B или A.update(B) — добавляет во множество A все элементы из множества B.

 A &= B или A.intersection\_update(B) — оставляет во множестве A только те элементы, которые есть во множестве B.

 A –= B или A.difference\_update(B) — удаляет из множества A все элементы, входящие во множество B.

 A ^= B или A.symmetric\_difference\_update(B) — записывает в A симметрическую разность множеств A и B.

 A <= B или A.issubset(B) — возвращает True, если A является подмножеством B.

 A >= B или A.issuperset(B) — возвращает True, если A является надмножеством B.

 A < B — эквивалентно A <= B and A != B.

 A > B — эквивалентно A >= B and A != B.

При использовании формы A.update(B) и аналогичных, параметр B может также являться списком, строкой и вообще любым итерируемым объектом. Так, запись A.update([1, 2, 1]) корректна (во множество будут добавлены элементы 1 и 2), а запись A |= [1, 2, 1] — породит исключение.

**15. Операции со словарями, учитывающие возможное отсутствие ключа. Операции многоэлементного изменения словарей. Операции поэлементного извлечения из словаря и их использование.**

Перед обращением к элементу мы можем проверять наличие ключа в словаре с помощью выражения ключ in словарь . Если ключ имеется в словаре, то данное выражение возвращает True :

Для удаления элемента по ключу применяется оператор del :

Cтоит учитывать, что если подобного ключа не окажется в словаре, то будет выброшено исключение KeyError .

Другой способ удаления представляет метод pop() . Он имеет две формы:

get(key) : возвращает из словаря элемент с ключом key . Если элемента с таким ключом нет, то возвращает значение None

get(key, default) : возвращает из словаря элемент с ключом key . Если элемента с таким ключом нет, то возвращает значение по умолчанию default

key = "+4444444"

if key in users:

user = users[key]

print(user)

else:

print("Элемент не найден")

users = {

"+11111111": "Tom",

"+33333333": "Bob",

"+55555555": "Alice"

}

del users["+55555555"]

print(users)

{ "+11111111": "Tom", "+33333333": "Bob"}

pop(key) : удаляет элемент по ключу key и возвращает удаленный элемент. Если элемент с данным ключом отсутствует, то генерируется исключение KeyError.

Если необходимо удалить все элементы, то в этом случае можно воспользоваться методом clear() :

Метод copy() копирует содержимое словаря, возвращая новый словарь:

Метод update() объединяет два словаря:

Для перебора словаря можно воспользоваться циклом for :

pop(key, default) : удаляет элемент по ключу key и возвращает удаленный элемент. Если элемент с данным ключом отсутствует, то возвращается значение default

users.clear()

users = {"+1111111": "Tom", "+3333333": "Bob", "+5555555": "Alice"}

students = users.copy()

print(students)

{"+1111111": "Tom", "+3333333": "Bob", "+5555555": "Alice"}

users = {"+1111111": "Tom", "+3333333": "Bob"}

users2 = {"+2222222": "Sam", "+6666666": "Kate"}

users.update(users2)

print(users)

{"+1111111": "Tom", "+3333333": "Bob", "+2222222": "Sam", "+6666666":

"Kate"}

users = {

"+11111111": "Tom",

"+33333333": "Bob",

"+55555555": "Alice"

}

for key in users:

print(f"Phone: {key} User: {users[key]} ")

При переборе элементов мы получаем ключ текущего элемента и по нему можем получить сам элемент.

Другой способ перебора элементов представляет использование метода items() :

Метод items() возвращает набор кортежей. Каждый кортеж содержит ключ и значении элемента, которые при переборе мы тут же можем получить в переменные key и value .

Кроме простейших объектов типа чисел и строк словари также могут хранить и более сложные объекты - те же списки, кортежи или другие словари:

В данном случае значение каждого элемента словаря в свою очередь представляет отдельный словарь.

Для обращения к элементам вложенного словаря соответственно необходимо использовать два ключа:

**16. Операции с представлениями словарей.**

Объекты, возвращаемые методами словарей dict.keys(), dict.values() и dict.items() являются объектами-представлениями (view). Они обеспечивают динамическое представление записей словаря. Другими словами, при изменении [словаря dict](https://docs-python.ru/tutorial/osnovnye-vstroennye-tipy-python/tip-dannyh-dict-slovar/) соответствующие представление будет отражать эти изменения.

#### len(dictview):

Функция [len(dictview)](https://docs-python.ru/tutorial/vstroennye-funktsii-interpretatora-python/funktsija-len/) вернет количество записей в словаре.

#### iter(dictview):

Функция [iter(dictview)](https://docs-python.ru/tutorial/vstroennye-funktsii-interpretatora-python/funktsija-iter/) возвращает [итератор](https://docs-python.ru/tutorial/osnovnye-vstroennye-tipy-python/tip-dannyh-iterator-iterator/) по ключам dict.keys(), значениям dict.values() или элементам dict.items(). В последнем случае представление будет в виде списка кортежей (key, value).

Итераторы представлений при добавлении или удалении записей в словаре могут вызвать ошибку RuntimeError или не выполнить итерацию по всем записям.

#### x in dictview:

Операция [x in dictview](https://docs-python.ru/tutorial/obschie-operatsii-posledovatelnostjami-list-tuple-str-python/prinadlezhnost-elementa-stroki-posledovatelnosti/) вернет, True если значение элемента x присутствует в представлении ключей, значений или элементов словаря. В последнем случае x должен быть кортежем (key, value).

#### reversed(dictview):

Функция [reversed(dictview)](https://docs-python.ru/tutorial/vstroennye-funktsii-interpretatora-python/funktsija-reversed/) вернет обратный итератор по ключам, значениям или элементам словаря. Представление будет повторяться в обратном порядке вставки.

#### dictview.mapping

Атрибут представления словаря [dictview.mapping](https://docs-python.ru/tutorial/operatsii-slovarjami-dict-python/dictview-mapping/) возвращает types.MappingProxyType, который обертывает исходный словарь, на который ссылается представление.

C dict.keys и dict.items можно использовать операции, присущие множествам

**17. Синтаксис и семантика обработки исключительных ситуаций в Python.**

Основные виды ошибок в программировании:

* синтаксические
* логические
* исключения

Синтаксические ошибки. Возникают, когда код нарушает правила языка программирования, такие как пропуск точки с запятой, незакрытые скобки или неправильный отступ кода.

Логические ошибки. Возникают, когда в коде реализована некорректная логика и код выдает неожиданный результат.

Исключения. Это ошибки, вызванные неожиданным условием, возникшим при выполнении кода. Это могут быть ссылки на несуществующие объекты, неверное число обязательных параметров функции, ошибки выхода за пределы массива и т. д.

Рассмотрим основные приемы и подходы исправления и обработки ошибок в программировании на Python.

В Python есть следующие базовые типы исключений:

От этих классов наследуются все конкретные типы исключений. В Python обладает довольно большим списком встроенных исключений. Весь этот список можно посмотретьв документации. Перечислю только некоторые наиболее часто встречающиеся:

BaseException : базовый тип для всех встроенных исключений

Exception : базовый тип, который обычно применяется для создания своих типов исключений

ArithmeticError : базовый тип для исключений, связанных с арифметическими операциями ( OverflowError , ZeroDivisionError , FloatingPointError ).

BufferError : тип исключения, которое возникает при невозможности выполнить операцию с буффером

LookupError : базовый тип для исключений, которое возникают при обращении в коллекциях по некорректному ключу или индексу (например, IndexError , KeyError )

IndexError : исключение возникает, если индекс при обращении к элементу коллекции находится вне допустимого диапазона

KeyError : возникает, если в словаре отсутствует ключ, по которому происходит обращение к элементу словаря.

OverflowError : возникает, если результат арифметической операции не может быть представлен текущим числовым типом (обычно типом float).

RecursionError : возникает, если превышена допустимая глубина рекурсии.

TypeError : возникает, если операция или функция применяется к значению недопустимого типа.

ValueError : возникает, если операция или функция получают объект корректного типа с некорректным значением.

ZeroDivisionError : возникает при делении на ноль.

NotImplementedError : тип исключения для указания, что какие-то методы класса не реализованы

Рассмотрим основные классы исключений:

* BaseException — является классом самого верхнего уровня;
* Exception — именно этот класс, а не BaseException, необходимо наследовать при создании пользовательских классов исключений;
* AssertionError — возбуждается инструкцией assert;
* AttributeError — попытка обращения к несуществующему атрибуту объекта;
* EOFError — возбуждается функцией input() при достижении конца файла;
* IOError — ошибка доступа к файлу;
* ImportError — невозможно подключить модуль или пакет;
* IndentationError — неправильно расставлены отступы в программе;
* IndexError — указанный индекс не существует в nоследовательности;
* KeyError — указанного ключа нет в словаре;
* KeyboardInterrupt — нажата комбинация клавиш <Ctrl+C>;
* NameError — попытка обращения к идентификатору до его определения;
* StopIteration — возникает, когда больше нет элементов, которые мог бы вернуть итератор;
* SyntaxError — синтаксическая ошибка;
* TypeError — тип объекта не соответствует ожидаемому;
* UnboundLocalError — внутри функции переменной присваивается значение после обращения к одноименной глобальной переменной;
* UnicodeDecodeError — ошибка преобразования последовательности байтов в строку;
* UnicodeEncodeError — ошибка преобразования строки в последовательность байтов;
* ValueError — переданный параметр не соответствует ожидаемому значению;
* ZeroDivisionError — попытка деления на ноль.

**18. Создание пользовательских исключений и инструкция assert.**

Инструкция *raise* имеет несколько форматов:

* raise <Экземпляр класса>
* raise <Имя класса>

В первом формате инструкции raise указывается экземпляр класса возбуждаемого исключения. При создании экземпляра можно передать данные конструктору класса.

try:

raise ValueError('Oпиcaниe исключения')

except ValueError as msg:

print(msg)

В другом случае указывается имя класса.

try:

raise ValueError

except ValueError:

print('Сообщение об ошибке')

Оператор assert в Python используется для отладки и написания тестов. Это утверждение, которое проверяет, является ли условие истинным. Если условие истинно, выполнение программы продолжается. Если условие ложно, оператор assert вызывает исключение AssertionError , что приводит к остановке выполнения программы.

**19. Модули в Python и их отличие от скриптов Python. Варианты синтаксиса импорта модуля и объектов модуля. Применение импортированных объектов. Порядок поиска модулей и специфика их загрузки.**

Модуль отличается от скрипта тем, что при импортировании в код скрипт выполнит действие, а модуль будет бездействовать до его применения.

Модуль Python – это файл, содержащий определения и операторы Python. Модуль может определять функции, классы и переменные. Модуль также может включать исполняемый код. Группировка связанного кода в модуль упрощает понимание и использование кода. Имя файла модуля совпадает с названием модуля, к которому добавлен суффикс .py.

Вариант синтаксиса импорта модуля и объектов модуля:

import module\_01

module\_01.fibo(10)

При импорте модуля с именем, например spam, интерпретатор сначала ищет встроенный модуль с таким же именем. Эти имена модулей перечислены в sys.builtin\_module\_names. Если он не найден, он ищет файл с именем spam.py в списке каталогов, заданном переменной sys.path. Первым расположением sys.path является каталог, содержащий скрипт модуля (текущий каталог).

import sys

sys.path

['D:\\Smirnov\\FinU\\2024-2025\\АиСД\_Python\\06\_Модули\_и\_пакеты',

'C:\\Users\\myfri\\anaconda3\\envs\\mvs\\python311.zip',

'C:\\Users\\myfri\\anaconda3\\envs\\mvs\\DLLs',

'C:\\Users\\myfri\\anaconda3\\envs\\mvs\\Lib',

'C:\\Users\\myfri\\anaconda3\\envs\\mvs',

'',

'C:\\Users\\myfri\\anaconda3\\envs\\mvs\\Lib\\site-packages',

'C:\\Users\\myfri\\anaconda3\\envs\\mvs\\Lib\\site-packages\\win32',

'C:\\Users\\myfri\\anaconda3\\envs\\mvs\\Lib\\site-packages\\win32\\lib',

'C:\\Users\\myfri\\anaconda3\\envs\\mvs\\Lib\\site-packages\\Pythonwin']

**20. Базовые операции для работы с файлами в Python. Использование инструкции with…as на примере работы с файлами.**

**21. Файлы с представлением информации в текстовом виде.**

Работа с текстовыми файлами в Python позволяет считывать данные из файлов, записывать данные в файлы и выполнять другие операции с файлами. В Python работа с текстовыми файлами осуществляется с использованием функций модуля io или операторов with и open. Вот основные операции, которые можно выполнить с текстовыми файлами в Python:

1. Открытие файла:

Для работы с текстовым файлом сначала нужно открыть его с помощью функции open(). Функция принимает имя файла и режим доступа в качестве параметров. Режимы доступа могут быть "r" (чтение), "w" (запись) или "a" (добавление).

2. Чтение данных из файла:

После открытия файла вы можете считать его содержимое с помощью метода read() или построчно с помощью метода readline().

Для чтения текстового файла используется функция open с аргументом "r" (read), а для записи или создания нового файла — функция open с аргументом "w" (write) или "a" (append) для дописывания данных в конец файла.

3.Запись данных в файл:

Пример записи в текстовый файл:

В этом примере мы создаем или открываем файл "file.txt" в режиме записи "w". При помощи метода write() мы записываем строку "Hello, World!" в файл.

При работе с текстовыми файлами можно также использовать другие методы, например:

 readline() — для чтения отдельных строк из файла;

 readlines() — для чтения всех строк из файла и возвращения их в виде списка.

Также важно помнить о закрытии файла после завершения работы с ним. В примерах выше оператор with автоматически осуществляет закрытие файла, но при использовании функции open без оператора with следует вызвать метод close() для закрытия файла.

**22. Модуль CSV, методы создания и работы с CSV файлами.**

Файлы формата CSV (Comma-Separated Values) являются текстовыми файлами, в которых значения разделяются запятыми. Работа с файлами CSV в Python осуществляется с помощью встроенной библиотеки csv. Вот основные операции, которые можно выполнить при работе с файлами CSV в Python:

1. Чтение данных из файла CSV:

Для чтения данных из файла CSV в Python используйте класс csv.reader. Он позволяет считывать данные построчно или как список значений.

2. Запись данных в файл CSV:

Чтобы записать данные в файл CSV, вам нужно открыть файл в режиме записи ("w" или "a") и использовать класс csv.writer. Он позволяет записывать данные строка за строкой или по списку значений. Вот пример записи данных в файл:

3. Работа с разделителями и кавычками:

В файле CSV могут использоваться различные разделители и символы кавычек. По умолчанию, в Python разделителем является запятая и кавычками — двойные кавычки. Однако, вы можете указать другие разделители и кавычки при необходимости, передав их в соответствующие аргументы функции csv.reader или csv.writer.

4. Работа с заголовками:

Если файл CSV содержит заголовки, вы можете использовать методы next() или \_\_next\_\_() для пропуска первой строки. Это позволит вам начать чтение или запись данных с первой фактической строки.

5. Обработка CSV-данных:

Считанные или записанные данные CSV обычно представляются в виде списков значений для каждой строки. Вы можете использовать индексирование или итерацию для обработки отдельных значений или строк в файле CSV.

6. Дополнительные возможности:

Библиотека csv предоставляет и другие полезные функции, такие как установка разделителя десятичной запятой, обработка NULL-значений, пропуск строк без данных и т.д. Подробности можно найти в документации Python.

При работе с файлами CSV важно учитывать особенности форматирования данных, такие как кавычки и экранирование символов, чтобы избежать проблем при чтении или записи.

**23. Модуль Pickle, создание и работа с бинарными файлами.**

Модуль pickle в Python предоставляет возможность сериализации (преобразования объектов Python в байтовую последовательность) и десериализации (восстановления объектов Python из байтовой последовательности) данных. Это позволяет сохранять сложные структуры данных, такие как списки, словари, классы и т.д., в файлы или передавать их по сети. Чтобы использовать модуль pickle, нужно импортировать его следующим образом: import pickle.

Сериализация объекта происходит при помощи функции pickle.dump(), которая принимает два аргумента: объект, который нужно сериализовать, и файловый объект, в который нужно записать сериализованные данные.

В этом примере мы создаем список data, который мы хотим сериализовать. Затем мы открываем файл "data.pkl" в режиме записи байтов ("wb") и с помощью функции pickle.dump() записываем сериализованные данные в файл. Для десериализации данных из файла используется функция pickle.load(), которая принимает файловый объект и возвращает восстановленный объект. Например:

В этом примере мы открываем файл "data.pkl" в режиме чтения байтов ("rb"). Затем мы используем функцию pickle.load(), чтобы загрузить данные из файла в переменную loaded\_data и выводим ее значение на экран.

Модуль pickle также предоставляет другие функции, такие как pickle.dumps() (сериализация в строку), pickle.loads() (десериализация из строки), pickle.dump() и pickle.load() для работы с сокетами.

Важно отметить, что модуль pickle может использоваться только в среде, где нет доверия небезопасным данных. При десериализации объектов из ненадежных источников может быть выполнен вредоносный код. Поэтому, будьте осторожны при использовании pickle с данными, полученными из ненадежных источников.

**24. Выражения генераторы и генераторы списков в Python. Использование условий в генераторах**.

Генерация списка:

lst\_gen = [el \* 2 for el in lst\_val]

Пример генератора списка с условием:

lst\_gen2 = [el \* 2 for el in lst\_val if el % 2 == 0]

Создание итерируемого объекта с помощью генератора на основе списка:

(x for x in lst\_val)

В результате создания выражения-генератора мы получили итерируемый объект. В отличие от генератора списков, здесь мы получили объект отложенных вычислений. То есть выражение-генратор знает, как выполнить действия для получения структуры данных, но пока эти действия не выполнены. Эти действия будут выполнены тогда, когда по итерируемому объекту начнется итерация, например при вызове функции next(). Такой подход позволяет эффективно использовать память.

Генерация множества:

sg1 = {х\*\*2 for х in [1, 2, 1, 2, 1, 2, 3]}

Генерация множества с условием:

{x for x in [1, 2, 1, 2, 1, 2, 3] if x % 2 == 0}

Генерация словарей:

Помимо генераторов списков и множеств язык Python поддерживает генераторы словарей. Синтаксис генераторов словарей похож на синтаксис генераторов списков, но имеет два отличия:

* выражение заключается в фигурные скобки, а не в квадратные;
* внутри выражения перед циклом for указываются два значения через двоеточие, а не одно:
  + значение, расположенное слева от двоеточия — ключ
  + значение, расположенное справа от двоеточия — значение.

Пример:

d18 = {k: v for (k, v) in [['a', 1], ['b', 2]]}

**25. Генераторы множества и словарей. Использование условий в генераторах.**

Генерация множества:

sg1 = {х\*\*2 for х in [1, 2, 1, 2, 1, 2, 3]}

Генерация множества с условием:

{x for x in [1, 2, 1, 2, 1, 2, 3] if x % 2 == 0}

Генерация словарей:

Помимо генераторов списков и множеств язык Python поддерживает генераторы словарей. Синтаксис генераторов словарей похож на синтаксис генераторов списков, но имеет два отличия:

* выражение заключается в фигурные скобки, а не в квадратные;
* внутри выражения перед циклом for указываются два значения через двоеточие, а не одно:
  + значение, расположенное слева от двоеточия — ключ
  + значение, расположенное справа от двоеточия — значение.

Пример:

d18 = {k: v for (k, v) in [['a', 1], ['b', 2]]}

**26. Анонимные функции.**

Анонимные функции в Python создаются с использованием ключевого слова lambda, за которым следует список аргументов в скобках, двоеточие и выражение, которое будет выполнено при вызове функции. Возвращаемое значение указывается после ключевого слова lambda. Рассмотрим примеры:

 функция, возвращающая квадрат числа:

square = lambda x : x\*\*2

 функция, складывающая два числа:

add = lambda a,b : a+b

 использование анонимной функции внутри другой функции:

def apply\_operation(x,y,operation)

return operation(x,y)

result = apply\_operation(4,2,lambda a,b:a\*b)

Анонимные функции полезны в случаях, когда не требуется создание именованных функций или когда нужна функция только для использования в одном месте программы.