МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКА КЫРГЫЗСКОЙ РЕСПУБЛИКИ

КОЛЛЕДЖ ЫССЫК-КУЛЬСКОЙ ГОСУДАРСТВЕННОЙ

УНИВЕРСИТЕТ ПО ИМЕНИ К.ТЫНЫСТАНОВА

СРС

**Тема: Основы Алгоритмизации**

Выполнил (а) : Муслимов Байдос Умарович

Проверил (а) : Эркинбаев Максат

Каракол 2023

Основные понятие алгоритмизации

1. Алгоритм - это последовательность инструкций, которые необходимо выполнить, чтобы решить определенную задачу. Алгоритм должен быть понятен человеку и выполним на компьютере.

2. Переменная - это область памяти, которая используется для хранения данных в программе. Переменная может содержать различные типы данных, такие как целые числа, дробные числа, строки и т.д.

3. Операторы - это инструкции, которые используются в алгоритмах для выполнения операций с данными. К ним относятся арифметические операторы (+, -, \*, /), операторы сравнения (>, <, =, !=) и логические операторы (and, or, not).

4. Циклы - это конструкции, которые позволяют выполнять определенный блок кода многократно. Они могут быть с условием (например, while) или без условия (например, for).

5. Условные операторы - это конструкции, которые позволяют выполнять определенный блок кода только если определенное условие истинно. К ним относятся if, elif и else.

6. Алгоритмическая сложность - это количество операций, которые нужно выполнить для решения задачи. Чем меньше алгоритмическая сложность, тем более эффективен алгоритм.

Понятие алгоритма. Свойства алгоритма.

1. Понятие алгоритма Алгоритм - это конечная последовательность шагов, направленных на достижение определенной цели. Он описывает процесс решения задачи и может быть использован для автоматизации повторяющихся операций. Алгоритмы являются ключевым инструментом в области информатики и программирования.

2. Свойства алгоритма 2.1 Корректность Алгоритм считается корректным, если он гарантированно решает поставленную задачу. Корректность может быть доказана математически или путем тестирования алгоритма на различных входных данных.

2.2 Определенность Алгоритм должен быть определенным, то есть каждый шаг должен быть ясно определен и понятен для исполнителя. Не должно быть неопределенных действий или пропусков.

2.3 Конечность Алгоритм должен завершаться после выполнения конечного числа шагов. Он не должен зацикливаться или бесконечно выполняться.

2.4 Входные данные Алгоритм должен быть определен для всех возможных входных данных. Он должен корректно обрабатывать все варианты входных данных.

2.5 Эффективность Алгоритм должен быть эффективным в том смысле, что он должен выполняться за приемлемое время и использовать приемлемое количество ресурсов.

3. Пример алгоритма Рассмотрим простой алгоритм нахождения наибольшего числа в массиве:

4. Задать начальное значение наибольшего числа равным первому элементу массива.

5. Для каждого элемента в массиве: а. Если текущий элемент больше наибольшего числа, заменить наибольшее число текущим элементом.

6. Наибольшее число в массиве найдено.

Данный алгоритм является корректным, определенным, конечным, корректно обрабатывает все входные данные и эффективен.

Формы записи алгоритмов

Алгоритм – это последовательность шагов, направленных на решение определенной задачи или достижение конкретной цели. Алгоритмы используются для автоматизации повторяющихся операций, и они являются ключевым инструментом в области программирования. Существует несколько форм записи алгоритмов, которые мы рассмотрим далее.

1. Естественный язык Алгоритм можно описать на естественном языке, таком как русский или английский. Такой способ записи удобен для понимания алгоритма человеком, но не является формой, пригодной для автоматизации и компьютерной обработки.

2. Блок-схемы Блок-схемы являются графическими представлениями алгоритмов. Они состоят из блоков, представляющих различные операции, и стрелок, показывающих направление выполнения алгоритма. Блоки могут быть различных форм и цветов, в зависимости от типа операции, которую они представляют. Например, прямоугольники могут представлять операции, ромбы – условия, а параллелограммы – ввод или вывод данных.

3. Псевдокод Псевдокод – это специальный язык программирования, который позволяет описывать алгоритмы в универсальной форме, независимо от конкретной платформы или языка программирования. Псевдокод представляет собой набор инструкций на высоком уровне абстракции, которые затем переводятся в конкретный язык программирования. Например, алгоритм нахождения наибольшего числа в массиве на псевдокоде может быть описан следующим образом:

1. Задать начальное значение наибольшего числа равным первому элементу массива.

2. Для каждого элемента в массиве:

а. Если текущий элемент больше наибольшего числа, заменить наибольшее число текущим элементом.

3. Вернуть наибольшее число.

4. Языки программирования Алгоритмы могут быть записаны на конкретных языках программирования, таких как C++, Python, Java и другие. Каждый язык программирования имеет свой синтаксис и правила написания, и алгоритма

Общие принципы построение алгоритмов

Построение алгоритмов – это процесс разработки последовательности шагов для решения задачи или достижения цели. Существует несколько общих принципов, которые могут помочь в создании эффективных и надежных алгоритмов.

1. Разбиение задачи на подзадачи Сложные задачи можно разбить на более простые подзадачи, что упрощает их решение. Этот принцип называется декомпозицией. Каждая подзадача может быть решена отдельно, что уменьшает сложность алгоритма и делает его более понятным.

2. Выбор подходящих структур данных Структуры данных – это способы организации и хранения информации в алгоритме. Выбор подходящей структуры данных может значительно ускорить выполнение алгоритма и снизить затраты на память. Например, использование хеш-таблиц может ускорить поиск элементов в списке, а использование бинарного дерева – ускорить сортировку.

3. Использование итераций и условий Итерации и условия позволяют выполнять определенные действия в зависимости от значений переменных или других условий. Итерации позволяют выполнять циклические действия, а условия – изменять порядок выполнения алгоритма в зависимости от определенных условий.

4. Управление памятью При выполнении алгоритмов необходимо управлять выделением и освобождением памяти. Некорректное управление памятью может привести к ошибкам и краху программы. Важно освобождать память после ее использования и не использовать больше памяти, чем это необходимо.

5. Обработка ошибок В алгоритмах всегда возможны ошибки, и необходимо предусмотреть их обработку. Важно определить, какие ошибки могут возникнуть, и как программа должна на них реагировать. Например, если программа не может открыть файл, то необходимо обработать эту ошибку и сообщить об этом пользователю.

6. Тестирование и оптимизация После написания алгоритма необходимо провести его тестирование и оптимизацию. Тестирование позволяет выявить ошибки и недочеты в работе алгоритма, а оптимизация – ускорить его выполнение. Оптимизация может включать

Линейные алгоритмы

Линейные алгоритмы - это класс методов машинного обучения, который используется для решения задач классификации и регрессии. Эти методы основаны на применении линейных функций для описания зависимости между входными и выходными данными. Они широко используются в индустрии и науке для решения задач в различных областях, таких как финансы, медицина, обработка естественного языка и многие другие.

Линейные алгоритмы имеют несколько преимуществ, которые делают их популярными в машинном обучении. Во-первых, они относительно просты в реализации и понимании, что делает их доступными для широкого круга людей. Во-вторых, они быстро работают на больших объемах данных, что делает их эффективными в решении задач в реальном времени. В-третьих, они легко интерпретируются, что позволяет понимать, какие функции наиболее важны для предсказания выходных данных.

В линейных алгоритмах используется модель, которая описывает зависимость между входными и выходными данными. Для задач классификации используется линейная модель, которая принимает на вход вектор признаков и возвращает метку класса. Для задач регрессии используется линейная модель, которая принимает на вход вектор признаков и возвращает числовое значение.

Наиболее распространенными линейными алгоритмами являются линейная регрессия и логистическая регрессия. Линейная регрессия используется для решения задачи регрессии и описывает линейную зависимость между входными и выходными данными. Логистическая регрессия используется для решения задачи классификации и описывает вероятность принадлежности входных данных к определенному классу.

Разветвляющийся алгоритмы.

Разветвляющиеся алгоритмы - это класс методов машинного обучения, которые используют древовидную структуру для принятия решений. Эти методы широко используются в машинном обучении для решения задач классификации, регрессии и кластеризации. Разветвляющиеся алгоритмы имеют множество преимуществ, которые делают их популярными в научном и промышленном сообществе.

В отличие от линейных алгоритмов, которые описывают линейную зависимость между входными и выходными данными, разветвляющиеся алгоритмы используют структуру дерева, чтобы разбить пространство признаков на подрегионы. Каждый узел дерева соответствует некоторому предикату на входных данных, который определяет, в какую ветвь дерева следует продолжить анализ данных.

Разветвляющиеся алгоритмы имеют несколько преимуществ, которые делают их популярными в машинном обучении. Во-первых, они могут обрабатывать данные с нелинейными зависимостями между входными и выходными данными. Во-вторых, они могут работать с данными, которые содержат различные типы признаков, включая числовые, бинарные и категориальные. В-третьих, они могут работать с данными, которые содержат отсутствующие значения.

Наиболее распространенными разветвляющимися алгоритмами являются деревья решений, случайные леса и градиентный бустинг деревьев решений. Деревья решений используются для решения задач классификации и регрессии и представляют собой дерево, в котором каждый узел представляет собой предикат на признаках данных. Случайные леса и градиентный бустинг деревьев решений представляют собой ансамбли деревьев решений, которые объединяются для повышения качества

Циклические алгоритмы.

Циклические алгоритмы - это класс методов, которые используют циклы для обработки данных. Эти методы широко используются в программировании для решения различных задач, таких как сортировка, поиск, обход и обработка данных. Циклические алгоритмы представляют собой эффективный и универсальный инструмент для решения множества задач, которые могут быть сформулированы в виде повторяющихся операций.

Циклические алгоритмы могут быть реализованы с помощью различных конструкций языков программирования, включая циклы for, while и do-while. Цикл for используется для выполнения заданного количества итераций, в то время как цикл while выполняется до тех пор, пока условие остается истинным. Цикл do-while является вариантом цикла while, который гарантирует выполнение хотя бы одной итерации.

Циклические алгоритмы имеют несколько преимуществ, которые делают их полезными в программировании. Во-первых, они позволяют автоматизировать повторяющиеся задачи, что может существенно сократить время и усилия, затрачиваемые на выполнение задач. Во-вторых, они могут обрабатывать данные различных типов и форматов, включая текстовые, числовые, бинарные и другие. В-третьих, они могут быть использованы для реализации различных алгоритмов и структур данных, таких как стеки, очереди, списки и т.д.

Наиболее распространенными циклическими алгоритмами являются алгоритмы сортировки, такие как сортировка пузырьком, сортировка вставками, сортировка выбором и быстрая сортировка. Они используют циклы для перебора элементов массива и выполнения операций сравнения и перестановки элементов. Другими примерами циклических алгоритмов являются алгоритмы поиска, такие как линейный поиск и двоичный поиск, а

Типы данных в Python.

Python - это интерпретируемый язык программирования, который имеет множество встроенных типов данных. В этом докладе мы рассмотрим основные типы данных в Python и их свойства.

1. Числовые типы данных Python поддерживает три типа числовых данных: целочисленные (int), числа с плавающей запятой (float) и комплексные числа (complex). Целочисленные числа - это целые числа, как положительные, так и отрицательные. Числа с плавающей запятой - это числа с десятичными знаками. Комплексные числа состоят из действительной и мнимой части.

2. Строковые типы данных Строки - это последовательности символов, заключенные в кавычки. В Python существует два типа строковых данных: обычные строки (str) и байтовые строки (bytes). Обычные строки используются для представления текстовых данных, а байтовые строки используются для представления бинарных данных.

3. Логический тип данных Логический тип данных (bool) может принимать только два значения: True и False. Логические значения используются для выполнения логических операций и условных выражений.

4. Списки, кортежи и словари Списки (list), кортежи (tuple) и словари (dict) являются коллекциями данных. Списки и кортежи представляют собой упорядоченные последовательности элементов, а словари представляют собой неупорядоченные наборы пар ключ-значение.

5. Множества и множества frozenset Множества (set) и множества frozenset - это коллекции уникальных элементов. Множества могут быть изменяемыми, а множества frozenset - неизменяемыми.

6. Неопределенные значения В Python есть два специальных значения, которые обозначают отсутствие значения: None и NaN. Значение None используется для обозначения отсутствия значения переменной, а NaN (Not a Number) используется для обозначения неопределенных или несуществующих чисел.

В заключение, в Python существует множество типов данных, каждый из которых предназначен для конкретного типа данных или задачи. Знание этих типов данных

Арифметические операции в Python.

Python поддерживает множество арифметических операций для работы с числами. В этом докладе мы рассмотрим основные арифметические операции в Python.

1. Операции сложения, вычитания, умножения и деления Операции сложения (+), вычитания (-), умножения (\*) и деления (/) являются основными арифметическими операциями в Python. Они выполняются над числами, как целочисленными, так и числами с плавающей запятой.

Примеры:

a = 10

b = 5

c = a + b # результат: 15

d = a - b # результат: 5

e = a \* b # результат: 50

f = a / b # результат: 2.0

2. Операция целочисленного деления и остатка от деления Операция целочисленного деления (//) возвращает результат деления, округленный до ближайшего меньшего целого числа. Операция остатка от деления (%) возвращает остаток от деления.

Примеры:

a = 10

b = 3

c = a // b # результат: 3

d = a % b # результат: 1

3. Операция возведения в степень Операция возведения в степень (\*\*), возводит число в указанную степень.

Примеры:

a = 2

b = 3

c = a \*\* b # результат: 8

Комбинированные операции Комбинированные операции позволяют выполнять арифметические операции и присваивание значения переменной в одной строке. Например, операция += выполняет сложение и присваивание значения.

Примеры:

a = 10

b = 5

a += b # эквивалентно a = a + b. Результат: 15

В заключение, в Python существует множество арифметических операций для работы с числами. Знание этих операций позволяет производить вычисления в программах на Python.

Биюлиотека (модуль) math.

Библиотека math в Python предоставляет множество математических функций, которые могут быть использованы в программах для выполнения различных вычислений. Для использования функций из библиотеки math, необходимо импортировать ее в программу.

Пример использования библиотеки math:

import math

# Вычисление квадратного корня

a = math.sqrt(25) # результат: 5.0

# Вычисление факториала

b = math.factorial(5) # результат: 120

# Вычисление значения синуса

c = math.sin(math.pi/2) # результат: 1.0

# Вычисление значения косинуса

d = math.cos(math.pi) # результат: -1.0

# Вычисление значения тангенса

e = math.tan(math.pi/4) # результат: 1.0

# Вычисление значения логарифма по основанию 2

f = math.log2(8) # результат: 3.0 Библиотека math также предоставляет множество других функций, включая тригонометрические функции, экспоненциальные и логарифмические функции, функции округления и др. Для получения полного списка функций, предоставляемых библиотекой math, можно посмотреть официальную документацию.

Библиотека math также предоставляет константы, такие как pi и e. Константы могут быть использованы в программах для выполнения вычислений.

Примеры:

import math

# Вычисление длины окружности

r = 5

length = 2 \* math.pi \* r # результат: 31.41592653589793

# Вычисление площади круга

area = math.pi \* r\*\*2 # результат: 78.53981633974483 В заключение, библиотека math предоставляет множество математических функций и констант, которые могут быть использованы в программах на Python для выполнения различных вычислений. Знание этой библиотеки позволяет эффективно использовать математические функции в программах и упростить процесс разработки.

Условный оператор в Python

Условные операторы - это инструменты в Python, которые позволяют программисту выполнить различный код в зависимости от выполнения определенного условия. В языке Python есть два основных условных оператора: if и elif.

Оператор if позволяет проверять условие и выполнить код, если условие истинно. Вот пример:

x = 5

if x > 0:

print("x is positive")

В этом примере, если значение переменной x больше нуля, то будет выведена строка "x is positive". Если условие не выполнено, то код в блоке if не будет выполнен.

Оператор elif используется для проверки дополнительных условий, если первое условие не было выполнено. Вот пример:

x = 5

if x > 0:

print("x is positive")

elif x < 0:

print("x is negative")

else:

print("x is zero")

В этом примере, если значение переменной x больше нуля, будет выведена строка "x is positive". Если значение переменной x меньше нуля, будет выведена строка "x is negative". Если значение переменной x равно нулю, будет выведена строка "x is zero". Если ни одно из условий не выполнено, будет выполнен блок кода else.

Также существует оператор if в одну строку, который может быть использован, если нужно выполнить простую проверку и выполнить одно действие в зависимости от результата. Вот пример:

x = 5

print("x is positive") if x > 0 else print("x is not positive")

В этом примере, если значение переменной x больше нуля, будет выведена строка "x is positive". Если значение переменной x не больше нуля, будет выведена строка "x is not positive".

Также операторы if и elif могут быть использованы вложенно, чтобы выполнить более сложные проверки. Вот пример:

x = 5

if x > 0:

if x < 10:

print("x is between 0 and 10")

else:

print("x is greater than 10")

else:

print("x is not positive")

В этом примере, если значение переменной x больше нуля, будет проверено, меньше ли значение x 10. Если это верно, будет выведена строка "x is between 0 and 10". Если значение x больше или равно 10, будет выведена строка "x is greater than 10". Если значение переменной x не больше нуля, будет выведена строка "x is not positive".

В заключение, условные операторы if и elif

Оператор цикла в Python

Операторы цикла в Python используются для повторения блока кода несколько раз. В языке Python существует два основных оператора цикла: for и while.

Оператор цикла for используется для повторения блока кода определенное количество раз. Вот пример:

for i in range(5):

print(i)

В этом примере, блок кода внутри оператора цикла for будет выполнен пять раз. Значение переменной i будет увеличиваться от 0 до 4 на каждой итерации.

Оператор цикла while используется для повторения блока кода до тех пор, пока определенное условие истинно. Вот пример:

x = 0

while x < 5:

print(x)

x += 1

В этом примере, блок кода внутри оператора цикла while будет выполнен до тех пор, пока значение переменной x меньше 5. На каждой итерации значение переменной x увеличивается на 1.

Внутри операторов цикла можно использовать операторы break и continue. Оператор break используется для прерывания цикла, если выполнено определенное условие. Вот пример:

for i in range(10):

if i == 5:

break

print(i)

В этом примере, блок кода внутри оператора цикла for будет выполнен до тех пор, пока переменная i меньше 5. Когда переменная i равна 5, оператор break прерывает выполнение цикла.

Оператор continue используется для пропуска текущей итерации цикла, если выполнено определенное условие. Вот пример:

for i in range(10):

if i % 2 == 0:

continue

print(i)

В этом примере, блок кода внутри оператора цикла for будет выполнен для каждого значения переменной i, кроме тех, которые делятся на 2. Оператор continue пропускает выполнение блока кода для таких значений переменной i.

Также операторы for и while могут быть использованы вложенно, чтобы выполнить более сложные циклы. Вот пример:

for i in range(3):

for j in range(2):

print(i, j)

В этом примере, блок кода внутри операторов цикла for будет выполнен шесть раз. Первый оператор цикла for будет выполнен три раза, а второй оператор цикла for - два раза.

Графика в Python

Python - мощный язык программирования, который может использоваться для создания различных графических приложений. Существует несколько библиотек, которые позволяют создавать графики в Python, таких как Matplotlib, Seaborn, Plotly и др.

Matplotlib - это наиболее популярная библиотека для создания графиков в Python. Она позволяет создавать различные типы графиков, такие как линейные графики, гистограммы, круговые диаграммы, точечные графики и многое другое.

Вот пример кода, который создает простой линейный график в Matplotlib:

import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]

y = [2, 4, 6, 8, 10]

plt.plot(x, y)

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.title('Linear graph')

plt.show()

В этом примере мы импортируем библиотеку Matplotlib и задаем значения переменных x и y. Затем мы используем функцию plt.plot(), чтобы создать линейный график. Функции plt.xlabel(), plt.ylabel() и plt.title() используются для добавления подписей к оси x, оси y и заголовка соответственно. Наконец, мы используем функцию plt.show() для отображения графика.

Seaborn - это еще одна библиотека для создания графиков в Python. Она предоставляет более продвинутые возможности для создания графиков, таких как графики плотности и ящики с усами. Она также может быть использована для создания более эстетически приятных графиков.

Вот пример кода, который создает гистограмму в Seaborn:

import seaborn as sns

tips = sns.load\_dataset("tips")

sns.histplot(tips["total\_bill"])

В этом примере мы импортируем библиотеку Seaborn и используем функцию sns.load\_dataset(), чтобы загрузить набор данных о чаевых. Затем мы используем функцию sns.histplot(), чтобы создать гистограмму распределения суммы счета.

Plotly - это библиотека для создания интерактивных графиков в Python. Она позволяет пользователю взаимодействовать с графиками, изменяя параметры и просматривая дополнительную информацию.

Анимация Графика в Python.

Python - мощный язык программирования, который может использоваться для создания анимаций графиков. Существует несколько библиотек, которые позволяют создавать анимации в Python, таких как Matplotlib, Plotly, Pygame и др.

Matplotlib - это наиболее популярная библиотека для создания анимации графиков в Python. Она позволяет создавать различные типы анимации, такие как линейные графики, гистограммы, круговые диаграммы, точечные графики и многое другое.

Для создания анимации графика в Matplotlib необходимо определить функцию, которая будет изменять данные на графике на каждом кадре анимации. Затем необходимо создать объект Figure и объект Axes, настроить начальное состояние графика и передать функцию в функцию FuncAnimation() для создания анимации. Функция FuncAnimation() используется для создания анимации. Функция update() используется для обновления данных на графике на каждом кадре анимации. Функция frames указывает количество кадров в анимации, а параметр interval указывает время задержки между кадрами. Параметр blit установлен в True для ускорения анимации.

Plotly - это библиотека для создания интерактивных графиков в Python. Она также может быть использована для создания анимации графиков.

Для создания анимации графика в Plotly необходимо определить начальное состояние графика, создать список кадров (frames), в каждом из которых определяется новое состояние графика, и передать список кадров в объект Figure.

Анимация графиков может быть использована для визуализации временных рядов, изменения параметров объектов, распространения волн и многих других процессов, которые происходят во времени.

Алгоритмы сортировки

Алгоритмы сортировки являются одними из наиболее изучаемых и важных алгоритмов в компьютерной науке. Они используются для упорядочивания данных в различных областях, таких как базы данных, поисковые системы, анализ данных и многих других.

Существует множество алгоритмов сортировки, но основными являются следующие:

1. Алгоритм сортировки пузырьком (Bubble sort) Этот алгоритм сортировки проходит по массиву несколько раз и на каждой итерации сравнивает два соседних элемента и меняет их местами,

если они расположены в неправильном порядке.

2. Алгоритм сортировки выбором (Selection sort) Этот алгоритм сортировки на каждой итерации ищет минимальный элемент в массиве и меняет его местами с первым элементом, затем он ищет второй минимальный элемент и меняет его местами со вторым элементом и так далее.

3. Алгоритм сортировки вставками (Insertion sort) Этот алгоритм сортировки проходит по массиву и вставляет каждый элемент на свое место в уже отсортированную часть массива.

4. Алгоритм сортировки слиянием (Merge sort) Этот алгоритм сортировки разбивает массив на множество подмассивов, каждый из которых состоит из одного элемента, затем он последовательно сливает эти подмассивы в отсортированный массив.

5. Алгоритм быстрой сортировки (Quick sort) Этот алгоритм сортировки выбирает элемент из массива (называемый опорным элементом) и перемещает все элементы, меньшие опорного, влево от него, а все элементы, большие опорного, вправо от него. Затем он рекурсивно повторяет этот процесс для каждого подмассива.

Каждый из этих алгоритмов имеет свои преимущества и недостатки в зависимости от конкретного случая использования. Выбор правильного алгоритма сортировки может ускорить выполнение программы и повысить ее эффективность.