

1) **Информатика** - наука, изучающая все аспекты получения, хранения, преобразования, передачи и использования информации. **Направления развития дисциплины** - теоретическая, прикладная, техническая.

2) **Информатика** - наука, изучающая все аспекты получения, хранения, преобразования, передачи и использования информации. **Структура информатики**: технических средств, программных средств, алгоритмических средств. Каковы задачи **теоретической информатики**? Построение фундамента Информатики как науки; обеспечение развития прикладной Информатики, практической Информатики и технической Информатики.

3) **Информация** – Любой вид знаний о предметах, фактах, понятиях и т. д. проблемной области, которыми обмениваются пользователи информационной системы. **Свойства информации**: объективность, достоверность, полнота, точность, актуальность, полезность. **Единица измерения информации** – бит (двоичный разряд), дит (десятичный разряд).

4) **Информация** – Любой вид знаний о предметах, фактах, понятиях и т. д. проблемной области, которыми обмениваются пользователи информационной системы. **Единица измерения информации** – бит (двоичный разряд), дит (десятичный разряд). **Формула Хартли** –  $H = k \log_a N$ .

5) **Информация** – Любой вид знаний о предметах, фактах, понятиях и т. д. проблемной области, которыми обмениваются пользователи информационной системы. **Единица измерения информации** – бит (двоичный разряд), дит (десятичный разряд). **Формула Шеннона** –  $I = -\sum_{i=1}^K p_i \log_2 p_i = -(p_1 \log_2 p_1 + p_2 \log_2 p_2 + \dots + p_K \log_2 p_K)$ ;  $H(X) = \sum_{i=1}^K p_i * \log(1/p_i)$ .

6) **Информация** – Любой вид знаний о предметах, фактах, понятиях и т. д. проблемной области, которыми обмениваются пользователи информационной системы. **Данные** – это формы представления информации, с которыми имеют дело информационные системы и их пользователи. Для того чтобы *информация стала данными*, необходимо провести процесс: сбор информации; формализации; кодирования; хранения данных.

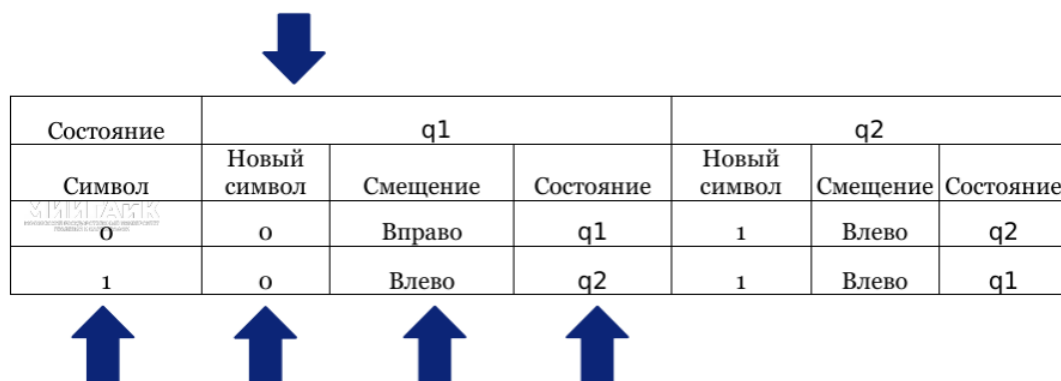
7) **Программирование** – Это процесс разработки и создания компьютерных программ с использованием специальных языков программирования. **Этапы программирования**: проектирование, написание кода, тестирования, отладка, оптимизация.

8) **Национальный проект "Цифровая экономика" Российской Федерации** — стратегическая инициатива, нацеленная на ускоренное развитие информационных технологий и цифровой трансформации различных сфер общества и экономики (ИИ, Big Data, виртуальная и дополнительная реальность, роботы, блокчейн, интернет вещей). **Цифровая трансформация**: представляет собой более глубокую и стратегическую трансформацию бизнес-модели и процессов компании с использованием цифровых технологий. **Этапы цифровой трансформации**: автоматизация, цифровизация, цифровая трансформация.

9) **Автоматизация** — первый шаг в этом направлении, обеспечивая эффективное использование технологий. **Этапы автоматизации:** планирование и подготовка; разработка и реализация; обучение и адаптация; масштабирование и оптимизация; эксплуатация и поддержка продукта.

10) **Машина Тьюринга** — это абстрактная машина (автомат), работающая с лентой отдельных ячеек, в которых записаны символы. Она состоит из трех частей: Лента, головка, управляющее устройство (УУ). Сначала идет символ алфавита, который должен быть записан в текущую ячейку  $aq$ , затем, указывается перемещение автомата влево (Л), вправо (П) или никуда (остаться на месте, Н). В конце указывается новое состояние, в которое должен перейти автомат  $qm$  ( $aq, L/R/S, qm$ ). **Алфавитом в машине Тьюринга** может быть любое конечное множество символов.

## Таблица переходов

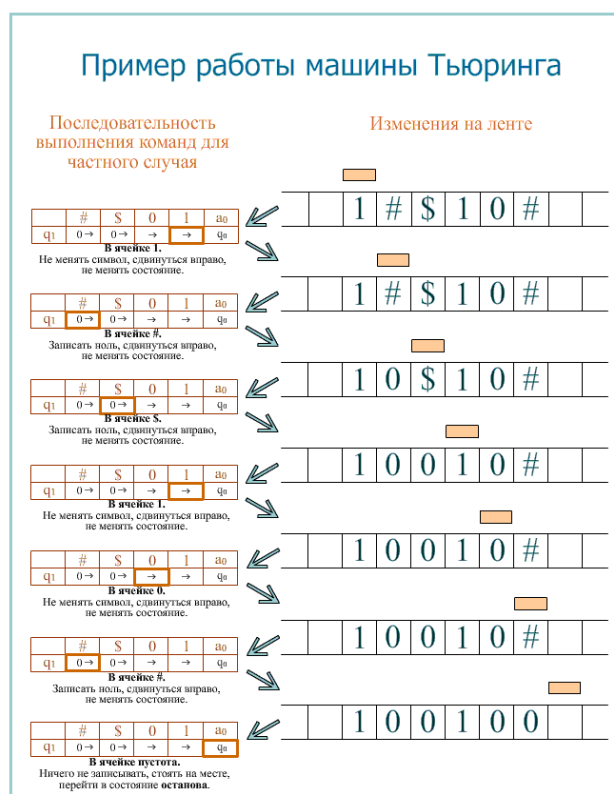


Состояние	q1			q2		
Символ	Новый символ	Смещение	Состояние	Новый символ	Смещение	Состояние
0	0	Вправо	q1	1	Влево	q2
1	0	Влево	q2	1	Влево	q1

11) **Машина Тьюринга** — это абстрактная машина (автомат), работающая с лентой отдельных ячеек, в которых записаны символы. Она состоит из трех частей: Лента, головка, управляющее устройство (УУ). **Алфавитом в машине Тьюринга** может быть любое конечное множество символов. **Система счисления** — это система письма для выражения чисел, то есть математическая нотация для представления чисел данного набора с использованием цифр или других символов последовательным образом.

12) **Система счисления** — это система письма для выражения чисел, то есть математическая нотация для представления чисел данного набора с использованием цифр или других символов последовательным образом. **Позиционная система** — значение каждой цифры зависит от её позиции (разряда) в числе. **Непозиционная** — каждая цифра числа имеет величину, не зависящую от её позиции (разряда). Пример позиционной системы: 2-я система счисления, 8-я система счисления, 10-я система счисления, 16-я система счисления. 2-я система счисления применяется в вычислительной технике. 16-я система счисления можно указать цвет, например: #FFFFFF — белый цвет, а #000000 — чёрный цвет, #FF0000 — красный цвет, #00FF00 — зелёный цвет, #0000FF — синий цвет. 8-я система счисления, как и двоичная, часто применяется в цифровой технике.

13) **Машина Тьюринга** — это абстрактная машина (автомат), работающая с лентой отдельных ячеек, в которых записаны символы. Она состоит из трех частей: Лента, головка, управляющее устройство (УУ).



14) **Алгоритм** — строго определенная последовательность действий для некоторого исполнителя, приводящая к поставленной цели или заданному результату за конечное число шагов. **Исполнитель** — субъект, способный исполнять некоторый набор команд. Совокупность команд, которые исполнитель может понять выполнить, называется **системой команд исполнителя**. **Входные данные** — те, что задаются для начала алгоритма. В результате выполнения алгоритма исполнитель должен получить искомый результат — **выходные данные**.

15) **Алгоритм** — строго определенная последовательность действий для некоторого исполнителя, приводящая к поставленной цели или заданному результату за конечное число шагов. **Исполнитель** — субъект, способный исполнять некоторый набор команд. **Свойства алгоритма:** дискретность (путь решения задачи разделён на отдельные шаги), понятность (алгоритм состоит из команд, входящие в системы команды исполнителя (СКИ)), определённости (команды понимаются однозначно), результативность (обеспечивается получение ожидаемого результата), массовость (обеспечивается решение задач с различными исходными данными).

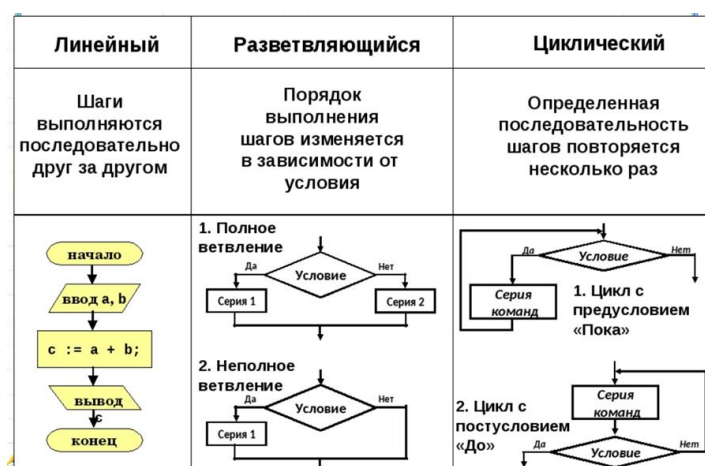
16) Наше воображение в целом допускает существование неразрешимых задач, то есть задач, для решения которых невозможно составить алгоритм. Исследованием таких задач занимается теория вычислимости. **Проблема останова** является алгоритмически неразрешимой.

17) **Тезис Чёрча — Тьюринга** — это гипотеза, постулирующая эквивалентность между интуитивным понятием алгоритмической вычислимости и строго формализованными понятиями частично рекурсивной функции и функции, вычислимой на машине Тьюринга. **Физический тезис Чёрча — Тьюринга**: любая функция, которая может быть вычислена физическим устройством, может быть вычислена машиной Тьюринга; **Сильный тезис Чёрча — Тьюринга**: любой конечный физический процесс, не использующий аппарат, связанный с непрерывностью и бесконечностью, может быть вычислен физическим устройством.

18) **Алгоритм** — строго определенная последовательность действий для некоторого исполнителя, приводящая к поставленной цели или заданному результату за конечное число шагов. **Способы представления алгоритмов**: словесная запись, блок-схема, формальные алгоритмические языки, псевдокод. **Псевдокод** — компактный (зачастую неформальный) язык описания алгоритмов, использующий ключевые императивных языков программирования, но опускающий несущественные подробности и специфический синтаксис.

19) **Алгоритм** — строго определенная последовательность действий для некоторого исполнителя, приводящая к поставленной цели или заданному результату за конечное число шагов. **Способы представления алгоритмов**: словесная запись, блок-схема, формальные алгоритмические языки, псевдокод. **Блок-схема** — при графическом представлении алгоритм изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий.

20) **Алгоритм** — строго определенная последовательность действий для некоторого исполнителя, приводящая к поставленной цели или заданному результату за конечное число шагов. **Виды алгоритмов**: линейный, разветвляющийся, циклический.



21) **Парадигма программирования** – совокупность идей и понятий, которые определяют общий стиль написания компьютерных программ, их структуры и отдельных элементов программной системы. **Цели парадигм программирования:** разделение программы на базовые составные элементы (напр., функции или объекты); определение модели преобразования данных; внедрение ограничений на используемые конструкции. **Парадигмы программирования** служат ориентиром в процессе разработки и позволяют разработчикам выбирать наиболее подходящие инструменты и подходы для конкретных задач.

22) **Парадигмы программирования:** императивное: процедурное (структурное), ООП; декларативное: логическое, функциональное. Некоторый язык программирования не обязательно использует только одну парадигму, многие языки поддерживают несколько парадигм, являясь мультипарадигменными; ни одна парадигма не может быть одинаково эффективной для всех задач, и программисту следует выбирать лучший стиль программирования для решения каждой отдельной задачи.

23) **Парадигма программирования** – совокупность идей и понятий, которые определяют общий стиль написания компьютерных программ, их структуры и отдельных элементов программной системы. **Императивное программирование** – парадигма, согласно которой программа представляет собой последовательность действий, изменяющих состояние программы. ("как" нужно сделать, а не "что" это делать). **Декларативное программирование** – парадигма, согласно которой программа представляет логику вычислений без описания прямой последовательности действий (действия определяются компилятором или интерпретатором). ("что" нужно сделать, а не "как" это делать).

24) **Парадигма программирования** – совокупность идей и понятий, которые определяют общий стиль написания компьютерных программ, их структуры и отдельных элементов программной системы. **Императивное программирование** – парадигма, согласно которой программа представляет собой последовательность действий, изменяющих состояние программы. ("как" нужно сделать, а не "что" это делать). **Состояние и изменение состояния:** Императивные языки позволяют программистам описывать изменения состояния программы, такие как изменение значений переменных или структур данных; **Последовательность выполнения:** В императивных языках программирования операции выполняются последовательно, в порядке, заданном программой. **Управление потоком:** Императивные языки предоставляют средства для управления потоком выполнения программы, такие как условные операторы (if-else), циклы (for, while) и переходы (break, continue, return); **Изменяемость данных:** В императивных языках программирования данные могут быть изменяемыми, и программист может явно изменять значения переменных и структур данных.

25) **Парадигма программирования** – совокупность идей и понятий, которые определяют общий стиль написания компьютерных программ, их структуры и отдельных элементов программной системы. **Структурное программирование** – парадигма, в основе которой лежит представление программы в виде иерархии блоков. **Теорема Бёма — Якопини** — положение структурного программирования, согласно которому любой исполняемый алгоритм может быть преобразован к структурированному виду, то есть такому виду, когда ход его выполнения определяется только при помощи трёх структур управления: последовательной, ветвлений и повторов или циклов.

26) **Парадигма программирования** – совокупность идей и понятий, которые определяют общий стиль написания компьютерных программ, их структуры и отдельных элементов программной системы. **Объектно-ориентированное программирование** – парадигма, согласно которой программа представляется в виде взаимодействующих объектов. **Основные концепции ООП**: объекты, классы, инкапсуляция, наследование, полиморфизм, абстракция. **Составляющее объекта**: идентификатор, свойства, методы.

27) **Парадигма программирования** – совокупность идей и понятий, которые определяют общий стиль написания компьютерных программ, их структуры и отдельных элементов программной системы. **Декларативное программирование** – парадигма, согласно которой программа представляет логику вычислений без описания прямой последовательности действий (действия определяются компилятором или интерпретатором). ("что" нужно сделать, а не "как" это делать). Подходы: математическое моделирование, искусственный интеллект, анализ данных, наука.

28) **Парадигма программирования** – совокупность идей и понятий, которые определяют общий стиль написания компьютерных программ, их структуры и отдельных элементов программной системы. **Логическое программирование** – парадигма, согласно которой программа представляет собой вывод с помощью правил формальной логики. **Основные концепции**: процедурная интерпретация правил, контроль над стратегией доказательства утверждений.

29) **Парадигма программирования** – совокупность идей и понятий, которые определяют общий стиль написания компьютерных программ, их структуры и отдельных элементов программной системы. **Функциональное программирование** – парадигма, согласно которой процесс исполнения программы представляется последовательностью вычислений значений для математических функций. **Основные концепции**: Чистые функции, Неизменяемость данных, Рекурсия, Высшие порядки функций, Высшие порядки функций, Ленивые вычисления. **Ленивые вычисления**: В функциональном программировании можно использовать ленивые вычисления, при которых вычисления выполняются только в тот момент, когда результат действительно требуется. Это может повысить производительность и уменьшить потребление ресурсов.

**30) Язык программирования** – Язык, предназначенный для представления программ. **Составляющие языка:** синтаксис, семантика, система выполнения и стандартная библиотека.

**31) Поколения языков (GL):** 1GL – **система команд процесса** (программа в машинном коде); 2GL – **машинно-ориентированные языки** (программы на ассемблере); 3GL – **процедурные языки** (FORTRAN, COBOL, C, Pascal, Basic), **структурное программирование**; 4GL – **объектно-ориентированные языки** (Java, C++, C#, Delphi, Smalltalk); 5GL – **современность**; декларативные языки (ориентированные на данные (SQL)), языки логического программирования (Prolog), предметно-ориентированные языки (DSL), событийные языки (Java, Perl, C#), сценарные языки (JavaScript, Perl, PHP, Python). **Ключевые цели и достижения:** 1GL – Первое поколение языков программирования было эффективным средством для написания программ в ранние дни компьютерной индустрии, но оно было трудоемким и требовало высокой квалификации. 2GL – Переносимость на уровне исходных кодов, Использование абстракций высокого уровня: Переменные, массивы; Операторы ветвлений, циклов, переходов; Функции, процедуры, подпрограммы; Работа с памятью; Библиотеки: работа с файлами, вводом/выводом и т.д. 3GL – **Процедурное программирование, Абстракция данных, Синтаксическая структура, Портбельность, Системные библиотеки.** 4GL – Языки этого поколения предназначены для реализации крупных проектов, повышают их надежность и скорость создания, *ориентированы на специализированные области применения*, и используют не универсальные, а объектно-ориентированные языки, оперирующие конкретными понятиями узкой предметной области. 5GL – новая технология производства микросхем, знаменующая переход от кремния к арсениду галлия, и дающая возможность на порядок повысить быстродействие основных логических элементов; новые способы ввода-вывода информации; отказ от традиционных алгоритмических языков программирования в пользу декларативных; ориентация на задачи искусственного интеллекта с автоматическим поиском решения на основе логического вывода.

**32) Транслятор** – это программ, которая транслирует программный код, написанный на одном языке, на другой язык. **Компилятор** – это программа, которая транслирует программный код, написанный на языке программирования в машинный код или ассемблер. **Интерпретатор** – это программ, которая воспринимает исходную программу на входном (исходном) языке и выполняет её. **Компилируемые ЯП:** C, C++, Fortran, C#, Java. **Интерпретируемые ЯП:** Perl, Schema, Python, JavaScript.

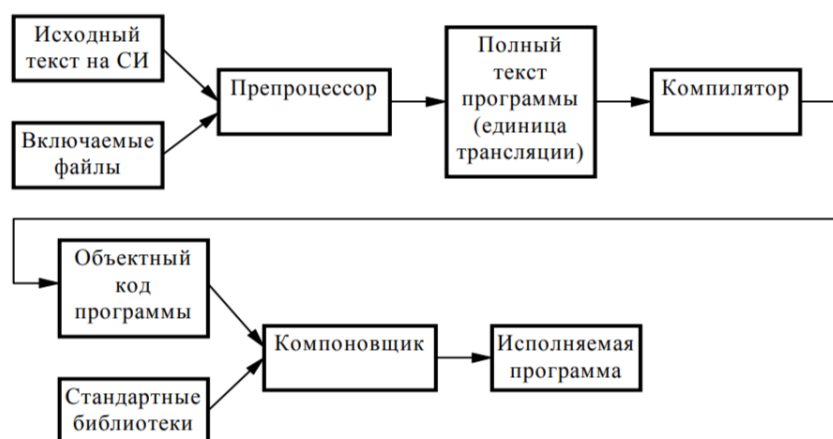
**33) Система типов языка программирования** - совокупность правил, определяющих свойство типа для конструкций языка (переменных, выражений, функций, модулей, ...). **Цели системы типов:** основная: определение интерфейсов для взаимодействия с частями программы и обеспечение их корректного использования с целью устранения ошибок; обеспечение функциональности языка (напр., динамическая диспетчеризация в ООП); рефлексия; оптимизация; повышение доступности программы для понимания. **Виды типизаций:** статическая (C++, Pascal, Java, C#) и динамическая (Python, PHP, Perl, JavaScript), сильная (Python, Java, C#) и слабая (C, C++, Visual Basic), явная (C, C++) и неявная (Python, Haskell).

34) За это десятилетие в мире родилось более тысячи разнообразных языков, как универсальных, так и специализированных, но выжили и доросли до XXI века дожили немногие, в том числе бессмертные Fortran, Basic, Algol, Cobol, Simula, Lisp и их потомки.

**Процедурные языки:** C, C++, C#, Java, Perl, Python, Fortran, Go.

35) **Smalltalk** — объектно-ориентированный язык программирования с динамической типизацией, основанный на идее посылки сообщений, разработанный в Xerox PARC Аланом Кэем, Дэном Ингаллсом, Тедом Кэглером, Адель Голдбергом, и другими в 1970-х годах. **Объектно-ориентированное программирование, Интерпретируемые языки, Интегрированные среды разработки (IDE), Графические пользовательские интерфейсы (GUI).**

36) **C** был разработан в 1972 году Деннисом М. Ритчи в исследовательском центре Bell Telephone Laboratories для создания операционной системы UNIX. **Структура и компоненты простой программы на C++:** исходный текст, препроцессинг, компиляция, компоновка, исполняемый файл.



37) **Java** была разработана с учетом потребностей развивающегося интернета, где сетевые и веб-приложения стали играть все более важную роль. Её ключевые черты и особенности стали идеально сочетаться с требованиями интернет-технологий: портируемость, сетевая ориентированность, безопасность, многозадачность. **JVM** — программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор. В 1995 г. фирма Sun Microsystems представила язык Java для программирования в интернете. Он возник в ходе реализации проекта Oak («Дуб»), целью которого было создание системы программирования бытовых микропроцессорных устройств.

38) **Python** — это высокоуровневый, интерпретируемый язык программирования, созданный Гвидо ван Россумом в начале 1990-х годов. **Дзен Python:** читаемость кода имеет значение; явное лучше, чем неявное; простота и ясность; сложность должна скрываться; плоский лучше, чем вложенный; один правильный способ сделать это; сейчас лучше, чем никогда; хотя никогда не бывает плохо, иногда лучше никогда; если реализацию сложно объяснить — это плохая идея; пространство имён — отличная штука, давайте делать их больше.



**Влияние других языков на Python:** ABC — отступы для группировки операторов, высокоуровневые структуры данных (map) (фактически, Python создавался как попытка исправить ошибки, допущенные при проектировании ABC); Modula-3 — пакеты, модули, использование else совместно с try и except, именованные аргументы функций (на это также повлиял Common Lisp); Си, C++ — некоторые синтаксические конструкции (как пишет сам Гвидо ван Россум — он использовал наиболее непротиворечивые конструкции из С, чтобы не вызвать неприязнь у Си-программистов к Python); Smalltalk — объектно-ориентированное программирование; Lisp — отдельные черты функционального программирования (lambda, map, reduce, filter и другие); Fortran — срезы массивов, комплексная арифметика; Miranda — списочные выражения; Java — модули logging, unittest, threading (часть возможностей оригинального модуля не реализована), xml.sax стандартной библиотеки, совместное использование finally и except при обработке исключений, использование @ для декораторов.