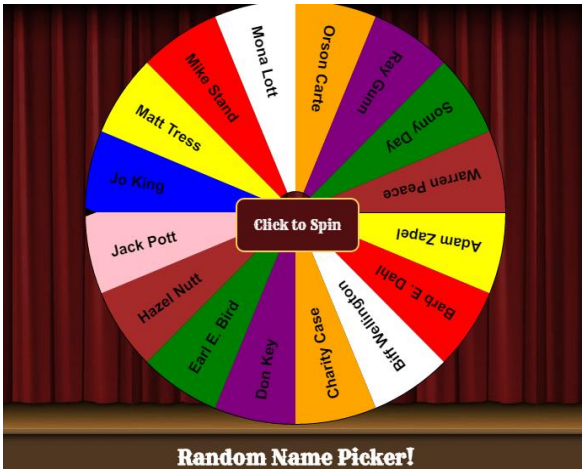
Школа:		
Дата:	ФИО учителя:	
Класс:	Участвовали:	Не участволвали:
Тема урока: Процедуры и функции - элементы структуризации программ		
Цели обучения, которые достигаются на данном уроке	Обучающая: ознакомить обучающихся с правилами оформления операторов управления, научиться использовать их при решении задач; Развивающая: развивать логическое мышление, внимание, память в процессе решения задач;	
Цели урока	**Все учащиеся смогут:** Определять разницу между процедурами и функциям **Большинство учащихся смогут:** Использовать и правильно оформлять операторов процедур и функции **Некоторые учащиеся смогут:** Научаться использовать их при решении программирований	
Критерии оценивания	Познакомятся с правилами оформления операторов управления. Научаться использовать их при решении задач	
Воспитание ценностей	Уважение к себе и к другим при постановке задач проекта через работу в парах и группе.	
Предварительные знания	Основные управляющие конструкции	
Межпредметные связи	информатика, математика, программирование	
Запланированные этапы урока	Запланированная деятельность на уроке	Ресурсы
Начало урока **5 мин**	Приветствие, проверка присутствующих. Объявление темы и целей урока. Делим детей на 3 группы при помощи игры Random  Актуализация знаний «Мозговой штурм». (5 минут). Метод «Толстые и тонкие вопросы» (для начала беседы по изучаемой теме).	

Середина урока 25мин	Долгое время процедуры и функции играли не только функциональную, но и архитектурную роль. Весьма популярным при построении программных систем был метод функциональной декомпозиции «сверху вниз», и сегодня еще играющий важную роль. Вся программа рассматривалась, как некоторая главная функция. В процессе проектирования программы происходила декомпозиция главной функции на подфункции, решающие частные задачи. Этот процесс декомпозиции продолжался до тех пор, пока не приходили к достаточно простым функциям, реализация которых не требовала декомпозиции и могла быть описана базовыми конструкциями языка программирования. с появлением ООП архитектурная роль функциональных модулей отошла на второй план. Для ОО-языков, к которым относится и язык C#, роль архитектурного модуля играет **класс**. Программная система строится из модулей, роль которых играют классы, но каждый из этих модулей имеют содержательную начинку, задавая некоторую абстракцию данных. Процедуры и функции связываются теперь с классом, они обеспечивают требуемую функциональность класса и называются методами класса. Поскольку класс в объектно-ориентированном программировании рассматривается как некоторый тип данных, то главную роль в классе начинают играть его данные – поля класса, задающие свойства объектов класса. Методы класса «служат» данным, занимаясь их обработкой. Помните, в C# процедуры и функции существуют только как методы некоторого класса, они не существуют вне класса. В данном контексте понятие класс распространяется и на все его частные случаи – структуры, интерфейсы, делегаты. В языке C# нет специальных ключевых слов – method, procedure, function, но сами понятия конечно же присутствуют. Синтаксис объявления метода позволяет однозначно определить, чем является метод – процедурой или функцией. Прежнюю роль библиотек процедур и функций теперь играют **библиотеки классов**. Библиотека классов FCL, доступная в языке C#, существенно расширяет возможности языка. Знание классов этой библиотеки, методов этих классов совершенно необходимо для практического программирования на C#, использование всей его мощи. **Процедуры и функции. Отличия** Функция отличается от процедуры двумя особенностями: • всегда вычисляет некоторое значение, возвращаемое в качестве результата функции; • вызывается в выражениях. Процедура C# имеет свои особенности: • возвращает формальный результат void, указывающий на **отсутствие результата,** возвращаемого при вызове процедуры; • вызов процедуры является оператором языка; • имеет **входные и выходные аргументы**, причем выходных аргументов – ее результатов – может быть достаточно много. Хорошо известно, что одновременное существование в языке процедур и функций в каком-то смысле избыточно. Добавив еще один выходной аргумент любую функцию можно записать в виде процедуры. Справедливо и обратное. Если допускать функции с побочным эффектом, то любую процедуру можно записать в виде функции. В языке C – дедушке C# – так и сделали, оставив только функции. Однако значительно удобнее иметь обе формы реализации метода – процедуры и функции. Обычно метод предпочитают реализовать в виде функции тогда, когда он имеет один выходной аргумент, рассматриваемый как результат вычисления значения функции. Возможность вызова функций в выражениях также влияет на выбор в пользу реализации метода в виде	

	функции. В других случаях метод реализуют в виде процедуры. **Описание методов (процедур и функций). Синтаксис** Синтаксически в описании метода различают две части – **описание заголовка** и **описание тела** метода: заголовок_метода тело_метода Рассмотрим синтаксис заголовка метода: [атрибуты][модификаторы]{void| тип_результата_функции} имя_метода([список_формальных_аргументов]) Имя метода и список формальных аргументов составляют **сигнатуру метода**. Заметьте, в сигнатуру не входят имена формальных аргументов, здесь важны типы аргументов. В сигнатуру не входит и тип возвращаемого результата. Квадратные скобки (метасимволы синтаксической формулы) показывают, что атрибуты и модификаторы могут быть опущены при описании метода. Подробное их рассмотрение будет дано в лекциях, посвященных описанию классов. Сейчас же упомяну только об одном из модификаторов – модификаторе доступа. У него четыре возможных значения, из которых пока рассмотрим только два – public и private. Модификатор public показывает, что **метод открыт** и доступен для вызова клиентами и потомками класса. Модификатор private говорит, что метод предназначен для внутреннего использования в классе и доступен для вызова только в теле методов самого класса. Заметьте, если модификатор доступа опущен, то по умолчанию предполагается, что он имеет значение private и метод является **закрытым** для клиентов и потомков класса. Обязательным при описании заголовка является указание типа результата, имени метода и круглых скобок, наличие которых необходимо и в том случае, если сам список формальных аргументов отсутствует. Формально тип результата метода указывается всегда, но значение void однозначно определяет, что метод реализуется процедурой. Тип результата, отличный от void, указывает на функцию. Вот несколько простейших примеров описания методов: void A() {…}; int B(){…); public void C(){…}; Методы A и B являются закрытыми, а метод C – открыт. Методы A и C реализованы процедурами, а метод B – функцией, возвращающей целое значение.	
Конец урока **10 мин**	К концу урока учащиеся научатся: Проведите работу по самооцениванию учащихся с помощью Лестницы успеха в рабочей тетради.	

Дифференциация – каким образом Вы планируете оказать больше поддержки? Какие задачи Вы планируете поставить перед более способными учащимися?	**Оценивание – как Вы планируете проверить уровень усвоения материала учащимися?**	**Охрана здоровья и соблюдение техники безопасности**
1. По уровню поддержки 2.По роли в групповой работе	1.Самооценивание по шаблону	Правила ТБ при работе в кабинете,

	2.Обратная связь по итогам выполнения заданий, по итогам рефлексии	Психологический комфорт
Рефлексия по уроку Была ли реальной и доступной цель урока или учебные цели? Все ли учащиеся достигли цели обучения? Если ученики еще не достигли цели, как вы думаете, почему? Правильно проводилась дифференциация на уроке? Эффективно ли использовали вы время во время этапов урока? Были ли отклонения от плана урока, и почему?	В планирование урока включены активные формы организации урока:	