**Git**

Git входит в минимум знаний программиста практически на любой специальности. Если максимально просто – это Google Documents для программистов. C помощью git вы сможете сохранять историю изменений вашей программы, дробить ее на отдельные версии, к примеру, у разных команд, и делать множество всего остального. Без Git разрабатывать продукт в команде практически невозможно.

Дальше нам нужно будет рассказать об основных сайтах, которые можно создавать удаленные репозитории - это **Github**,**Gitlab**, **Bitbucket**.

На каждый из репозиториев нужно привести свою небольшую характеристику. И если что, git c шведского и со сленга английского переводится как "мерзавец" ( это типа прекол такой от Линус Торвальдса, когда он разрабатывал GIT), но студенты должны сами узнать, что это такое.

Окей, теперь перейдем к ресурсам - Github вне сомнения, самый популярный из всех предложенных сервисов. Раньше он был «на страже опенсорса», но сейчас оказался в руках Microsoft, и пока они не успели там ничего сломать. Там можно искать интересные проекты, библиотеки, есть некоторый аналог умный ленты из фейсбука, в котором вам рекомендуют популярные репозитории

**Gitlab**

Больше популярен у небольших команд из-за более выгодного тарифа на приватные репозитории и таких фишек, как CI/CD (система непрерывного развертывания проекта), есть и другие плюсы, такие как более выгодные коммерческие тарифы, роли в проектах, собственная вики-система, и прочее

**Bitbucket**

Bitbucket - одна из небольших платформ. Немного медленная работа в России, но есть свои плюсы - действительно простой интерфейс, возможность заводить аккаунт от Google, RSS - лента, и прочие радости жизни.

Здорово, с самими сервисами мы закончили, теперь можно и посмотреть на программу Git, с которой нам и предстоит работать для того, чтобы сохранять наши изменения.

Для чего вообще нужен **GIT**? Это система управления версиями. Вместо того, чтобы делать файл с названиями "версия от 14 числа", можно попросить систему управления делать "слепки" каждый раз, когда вам нужно будет. Сделали какое-то важное достижение и не хотите, чтобы это потерялась - вы делаете слепок. Если вдруг что-то сломается, или вы захотите откатиться назад - git предоставляет эти возможности. Если вы захотите сделать "версию проекта для других целей", вы вполне можете сделать отдельную ветку. Ветки мы еще пока не проходили, и сегодня будем работать только в одной ветке, которая сразу создается по умолчанию - master. Она же главная ветка. Вообще в ней стараются писать только максимально оттестированный код, который представляет из себя «максимально оттестированный» однако это правило соблюдается далеко не всегда.

У нас есть несколько основных команд, без которых создание любого git - репозитория просто невозможно:

***git init*** - инициализация git репозитория. После этой команды у вас создается скрытая папка .git, в котором у нас записываются все изменения.

***git add --all -***добавление файлов в слежение. Чаще всего добавляются все файлы подряд, чтобы упростить процесс разработки. Те файлы и папки, которые слишком тяжелые и приватные, можно добавить в игнорирование с помощью файла .gitignore

***git commit -m "message about your changes"*** - изготовление "слепка" ваших изменений. После флага -m идет только одно сообщение, которое потом может пригодиться для отслеживания, что было сделано в этом коммите.

***git remote add origin <адрес удаленного репозитория>*** - подключения удаленного репозитория к конкретному проекту.

После того, как у нас будет подключен удаленный репозиторий, мы сможем туда заливать наш код. Он у нас имеет короткий псевдоним origin, который у нас может быть любым названием, просто первый традиционно называется origin. Кроме того, вы можете подвязывать другие удаленные репозитории, если вам будет нужно.

Кроме того, можно предупредить студентов, чтобы они не создавали Readme.md, потому что это может сломать им push и тогда придется ковыряться с pull.

Кроме того, вначале студентам придется зарегистрироваться внутри git. Для этого нужно будет указать свою почту и имя, совершенно не обязательно настоящие, однако иначе он не разрешит пушить. Кроме того, придется ввести и данные с github, но это гит должен сам спросить в то время, когда мы попробуем провести операцию, которая требует каких-либо прав.

После этого студенты могут сами попробовать залить любой проект (к примеру то, что они сделали на прошлом уроке). Много на это времени не тратим, если очень сильно не получается - пускай лучше сделают это на уроке.

**Pycharm Community Edition**

Сегодня мы еще посмотрим на Pycharm. Про Pycharm мы уж рассказывали в прошлый раз, пришло время рассказать немного подробнее про него. Pycharm был построен на основе IntelliJ IDEA. Community обладает, конечно, немного усеченными возможностями (там нет настолько большой поддержки веб-языков, развертывания своего виртуального сервера, поддержка большего количество синтаксиса фреймворков). Но в общем, для только начала программирования, CE версии нам будет достаточно.

Нужно сделать несколько основных вещей - это настроить базовый интерпретатор для того, чтобы у нас вообще могла работать программа, и открыть настройки, чтобы студенты могли освоиться с такими базовыми настройками, как настраивание шорткатов и установка плагинов. Можно попросить студентов самостоятельно поставить какой-либо из плагинов, чтобы они могли освоиться с базовой настройкой этого инструмента. Кроме того, можно поставить прекрасное фоновое изображение интернет-магазина Ozon.

Поговорим об основных настройках программы Pycharm, чтобы. Их можно было бегло обсудить со студентами:

**Appearance** – внешний вид редактора, темная/ светлая тема, выбор шрифтов, заднего фона, размер шрифтов

**Keymap** – настройка своих шорткатов для работы с кодом, таких операций как дублирование строк, увеличение/уменьшение шрифта кода, и множество всего полезного.

**Editor –** настройки редактора, его внешний вид, дефолтное поведение, наборы файлов, которые редактор может обрабатывать

**Plugins -**  местный магазин плагинов. Плагины отвечают за различные части внешнего вида, либо за знание синтаксиса какого-то языка, такого, к примеру, как Bash, или какой-либо библиотеки css. Из полезного можно поставить к примеру Material Design.

**Version Control -**  для работы с системами контроля, такими к примеру, как Git

***Списки, циклы и начало условных конструкций Python***

Теперь мы можем заняться Python. Сегодня мы начинаем проходить работу с массивами (списками). Говорим, что сначала списки кажутся достаточно бесполезными, однако в реальном программировании, к примеру процедурном (но до этого момента ещё нужно дожить).

Можно создавать массивы с любыми данными внутри по типу переменных, к примеру, вполне легально создавать массив массивов.

Пока для простоты мы вполне можем создать массив, который наполнен строчными типами переменных, и посмотрим основные методы работы с массивами:

<code>

cars.append(‘jigul’) – добавить новый объект в имеющийся массив

cars.reverse() – изменить порядок элементов на обратный

count(cars) – количество элементов в массиве

del cars[0] - удалить элемент

cars.remove(‘volvo’) – удалить первое вхождение аргумента в списке

cars.pop(0) – удаляем 0 элемент из массива и возвращаем его.

</code>

После этого мы можем посмотреть на работу с сортировкой данных. Все тот же массив мы можем отсортировать как правильно, так и неправильно.

<code>

cars.sort() – «правильная» сортировка по алфавиту

cars.reverse() – изменить порядок элементов на обратный

</code>

После предыдущего занятия у нас должен остаться код следующего содержания:

<code>

print("Здраствуйте! Программа по подсчету обших трат")

salary\_arr = []

name1 = input("Данные первого человека: ")

name2 = input("Данные второго человека: ")

name3 = input("Данные третьего человека: ")

# вызывать каждый раз какую то функцию и класть в массив эти данные, уже обработанные

salary1 = input("Доход первого человека")

salary2 = input("Доход второго человека")

salary3 = input("Доход третьего человека")

amount = int(salary1) + int(salary2) + int(salary3)

mean = amount / 3

mean = str(mean)

print(name1.title() + " Может тратить: " + mean)

print(name2.title() + " Может тратить: " + mean)

print(name3.title() + " Может тратить: " + mean)

</code>

Теперь мы его немного трансформируем для оптимизации в массивы

<code>

print("Здраствуйте! Программа по подсчету обших трат")

data\_arr = []

salary\_arr = []

data\_arr.append(input("Данные первого человека: "))

data\_arr.append(input("Данные второго человека: "))

data\_arr.append(input("Данные третьего человека: "))

# вызывать каждый раз какую то функцию и класть в массив эти данные, уже обработанные

salary\_arr.append(input("Доход первого человека"))

salary\_arr.append(input("Доход второго человека"))

salary\_arr.append(("Доход третьего человека"))

amount = int(salary\_arr[0]) + int(salary\_arr[1]) + int(salary\_arr[2])

mean = amount / 3

mean = str(mean)

print(data\_arr[0].title() + " Может тратить: " + mean)

print(data\_arr[1].title() + " Может тратить: " + mean)

print(data\_arr[2].title() + " Может тратить: " + mean)

</code>

Так намного короче, не правда ли?

Но сами по себе списки довольно бесполезны, они куда полезнее в связке с циклами и элементом рандом. Давайте посмотрим на работу цикла for range и попробуем улучшить нашу программу( ведь это даст дополнительно сокращение)

<code>

data\_arr = []

salary\_arr = []

amount = 0

#создадим пустые массивы и пустую переменную для подсчету общего дохода участников процесса

for i in range(3):

data\_arr.append(input("Имя " +str(i+1) +" человека: "))

# вызывать каждый раз какую то функцию и класть в массив эти данные, уже обработанные

for i in range(3):

salary\_arr.append(input("Доход " + str(i+1) + " человека"))

for salary in salary\_arr:

amount += int(salary)

mean = amount / 3

mean = str(mean)

for name in data\_arr:

print(name.title() + " Может тратить: " + mean)

</code>

Как вы видите, наш код уменьшился и стал куда более изящным. Теперь давайте перейдем к следующему вопросу - работу с библиотекой random

библиотека random поставляется вместе с Python, но ее надо проимпортировать в программу (а то она излишне раздувала бы размер оперативной памяти, если бы была бы просто сразу в составе интерпретатора программы). В модуле random есть множество полезных функций, к примеру,

random.random() - выдает случайное дробное число от 0 до 1

random.randint(1, 10) - возвращает случайно число от 1 до 10. Требуем от студентов, чтобы они немного попробовали поиграться с другими числами

random.shuffle(cars) - перетасовка массива как «колоды карты». Довольно полезная перестановка, которую в других языках надо писать "ручками". Может быть использована для написания всякого рода смешных программ. Например, давайте напишем различные варианты русской рулетки:

random.choice(cars) - случайный выбор элемента из массива.

Давайте напишем небольшую лотерею:

<code>

import random

guess\_number = input("Выберите любое число: ")

limit = input("введите пределе рандома: ")

limit = int(limit)

random\_number = random.randint(0, limit )

if random\_number == int(guess\_number):

print("могу вас поздравить, вы выйграли!")

else:

print("сегодня вам не везет")

print("выйгрышное число на самом деле", random\_number)

</code>

Отлично, это у нас программа несложной лотереи, в ней можно попробовать свою удачу. Однако мы теперь можем продолжить размышлять, как вообще работает random. Можно спросить у студентов, как они представляют себе, как можно написать программу русской рулетки, потому что мне эта программа нравится тем, что есть множество вариантов, как это можно написать и реализовать

Программа русской рулетки:

<code>

# к примеру:

import random

amount\_of\_bullets = input('Сколько патронов вы собираетесь вставить в револьвер?')

aob = int(amount\_of\_bullets)

baraban = [0,0,0,0,0,0]

for i in range(aob):

baraban[i] = 1

print("Посмотрите на барабан", baraban)

How\_much = input("сколько раз вы собираетесь нажимать на курок?")

hm = int(How\_much)

for i in range(hm):

random.shuffle(baraban)

print(baraban[0])

</code>

Программу можно дополнить, если после каждой раскрутки она будет переспрашивать, хотите ли вы еще поиграть, и если хотите, тогда она запустится. Это вполне можно перезаложить к примеру в цикл.