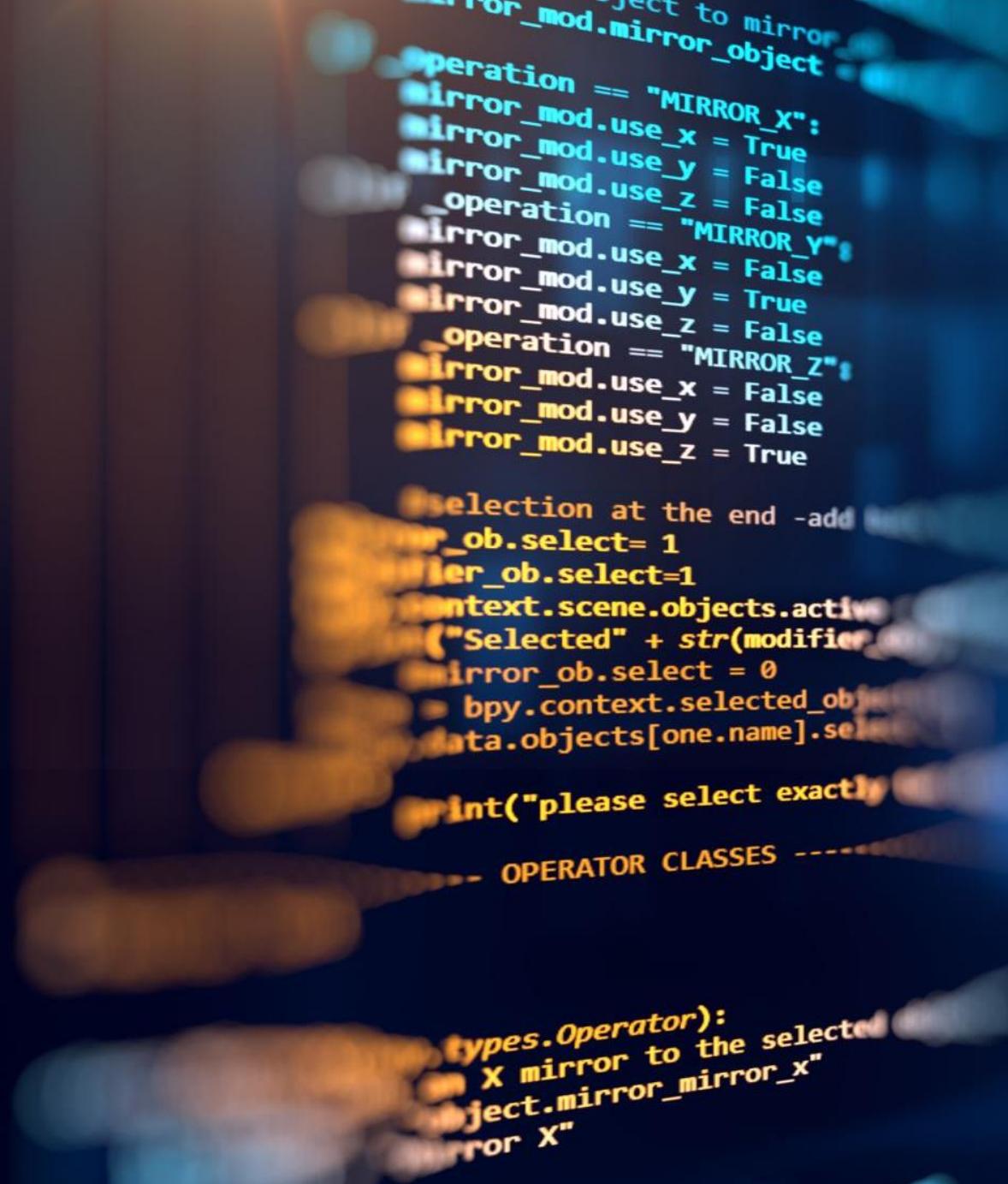


# Языки прикладного программирования

Лекция 2



```
    for object in objects:
        if object == selected_object:
            mirror_mod.mirror_object = object
            mirror_mod.use_x = True
            mirror_mod.use_y = False
            mirror_mod.use_z = False
        else:
            mirror_mod.mirror_object = None
            mirror_mod.use_x = False
            mirror_mod.use_y = False
            mirror_mod.use_z = False

    if len(objects) > 1:
        print("please select exactly one object")
        return {"FINISHED"}  
  
- OPERATOR CLASSES -  
  
class Operator(bpy.types.Operator):
    bl_idname = "object.mirror"
    bl_label = "X mirror to the selected object"
    bl_options = {'REGISTER', 'UNDO'}
```

Что осталось  
непонятным из  
материала 1 лекции?

ООП в Python

# Основной синтаксис

```
class class_name:

    static_field = 42 # аналог static из .NET/C++ доступно без создания класса
    __private_static_field = 42 # "приватне" поле, но доступ все равно можно получить
    # __class_name__private_static_field

    @staticmethod # декоратор обозначающий что метод статический
    def static_method(): # статический метод
        print("static method")

    def __init__(self): # конструктор объекта
        # self - аналог this, должен быть в каждом методе класса
        self.class_field = "some str" # публичное поле класса
        self.__private_field = "secret" # приватное поле класса

    def method(self, param): # метод класса
        return self.__private_field + " " + param

print(class_name.static_field) # печать статического поля
class_name.static_method() # вызов статического метода

obj = class_name() # создание класса
print(obj.class_field) # доступ к полю класса
obj.method("some value") # вызов метода
```

# Свойства

```
# пример применения свойств:
class Rectangle:

    def __init__(self, width, height):
        self.__width = width
        self.__height = height

    @property
    def width(self):
        return self.__width

    @width.setter
    def width(self, w):
        if w > 0:
            self.__width = w
        else:
            raise ValueError

    @property
    def height(self):
        return self.__height

    @height.setter
    def height(self, h):
        if h > 0:
            self.__height = h
        else:
            raise ValueError

    def area(self):
        return self.__width * self.__height

rect = Rectangle(10, 20)
print(rect.width)
print(rect.height)
```

# Наследование

- ❖ Доступно множественное наследование
- ❖ Все классы унаследованы от object, явно это указывать не нужно
- ❖ Синтаксис создания класса наследника:
  - ❖ `class имя_класса(имя_родителя1, [имя_родителя2,..., имя_родителя_n])`
- ❖ super – ключевое слово, для обращения к родительскому классу из наследника
- ❖ isinstance(object, type) – проверка принадлежит ли объект типу (точное совпадение)

# Пример

```
class Figure:  
    def __init__(self, color):  
        self.color = color  
  
    # пример переопределения метода (переопределяется метод object)  
    # __str__ отвечает за строковое представление метода  
    def __str__(self):  
        print("Figure")  
        print("Color: " + self.__color)  
  
class Rectangle(Figure):  
    def __init__(self, width, height, color):  
        super().__init__(color) # вызов базового конструктора  
        self.width = width  
        self.height = height  
  
    def area(self):  
        return self.width * self.height  
  
    # пример переопределения метода (переопределяется метод object)  
    def __str__(self):  
        print("Rectangle")  
        print("Color: " + self.color)  
        print("Width: " + str(self.width))  
        print("Height: " + str(self.height))  
        print("Area: " + str(self.area()))
```

# Итераторы

- ❖ Итератор – объект который реализует метод `__next__` и позволяет обходить коллекцию
- ❖ Окончание обозначается при помощи бросания исключения `StopIteration`
- ❖ Чтобы по классу можно было итерироваться необходим метод `__iter__` возвращающий итератор
  - ❖ (*в примере объект сам себе итератор*)

```
class SimpleIterator:  
    def __iter__(self):  
        return self  
  
    def __init__(self, limit):  
        self.limit = limit  
        self.counter = 0  
  
    def __next__(self):  
        if self.counter < self.limit:  
            self.counter += 1  
            return self.counter  
        else:  
            raise StopIteration  
  
s_iter = SimpleIterator(5)  
for i in s_iter:  
    print(i)
```

Подробнее о функциях

# Что такое функция в python

- ❖ Функция в python это объект с методом `def __call__(self)`
- ❖ Функция может создаваться в другой функции
- ❖ Функции могут передаваться в другие функции как параметры

# Lambda

- ❖ Lambda-функция – это безымянная функция с произвольным числом аргументов и вычисляющая одно выражение. Тело такой функции не может содержать более одной инструкции (или выражения).

```
l = [1, 2, 3, 4, 5, 6, 7]
f = lambda x: x**3
print(list(map(f, l)))

## [1, 8, 27, 64, 125, 216, 343]
```

# Замыкания

- ❖ Замыкание - функция, которая находится внутри другой функции и ссылается на переменные объявленные в теле внешней функции.

```
# пример функции которая возвращает функцию (замыкание) умножения на переданный параметр a
def mul(a):
    def helper(b):
        return a * b
    return helper

two_mul = mul(2)
print(two_mul(5)) # 10
```

# Декораторы

- ❖ Декоратор – функция, аргументом которой является другая функция.
- ❖ Декоратор предназначен для добавления дополнительного функционала к функции без ее изменения.

```
# декоратор для функции с одной переменной
def print_fn_info(fn):
    def wrapper(arg):
        print("Run function: " + str(fn.__name__) + "(), with param: " + str(arg))
        return fn(arg)
    return wrapper
```

# Примеры

```
def pow2(x):
    return x * x

print(pow2(2))

weapped_pow2 = print_fn_info(pow2)
pow2 = weapped_pow2

print(weapped_pow2(2)) # Run function: pow2(), with param: 2
print(pow2(2)) # Run function: pow2(), with param: 2

# чтобы не писать постоянно

# weapped_pow2 = print_fn_info(pow2)
# pow2 = weapped_pow2

# используется следующий синтаксис:

@print_fn_info
def pow2(x):
    return x * x
```

# Примеры

```
# декоратор для метода класса
def method_decor(fn):
    def wrapper(self):
        print("Run method: " + str(fn.__name__))
        fn(self)
    return wrapper

class Vector():
    def __init__(self, px = 0, py = 0):
        self.px = px
        self.py = py

    @method_decor
    def norm(self):
        print((self.px**2 + self.py**2)**0.5)

vc = Vector(px=10, py=5)
print(vc.norm())
# Run method: norm
# 11.180339887498949
```

# Модули и пакеты

# Модули

- ❖ Модуль – файл с расширением .py\*

- ❖ Подключение модуля:

- ❖ *import <module\_name1>, <module\_name2>, <module\_name3>*
- ❖ *import <module\_name1> as <new\_name>*
- ❖ *from <module\_name> import <name1>, <name2>*
- ❖ *from <module\_name> import \**
- ❖ *from <module\_name> import <name> as <new\_name>*

- ❖ \* - Модуль может быть написан не только на python

```
from math import factorial as f
f(4)

from math import cos, sin, pi
cos(pi/3)
sin(pi/3)

import math as m
m.sin(m.pi/3)

from math import *
cos(pi/2)
sin(pi/4)
```

# Пакеты

- ❖ Пакет – каталог, содержащий файл `__init__.py`.
- ❖ Пакеты используются для формирования пространства имен, что позволяет работать с модулями через указание уровня вложенности (через точку).
- ❖ Пакет может включать в себя модули и другие пакеты.
- ❖ Для импортирования пакетов используется тот-же синтаксис, что и для импорта модулей.

# Пакеты

- ❖ Файл `__init__.py` может быть пустым или содержать переменную `__all__`, хранящую список модулей, который импортируется при загрузке
- ❖ Пример – см. репозиторий (`Python/my_package`)

# Установка сторонних пакетов

- ❖ Сторонние пакеты устанавливаются при помощи утилиты pip (pip3 если у вас установлен python2)
  - ❖ Pip install \_имя\_пакета\_
- ❖ Для изоляции версий могут быть использованы различные виртуальные окружения (подробности [по ссылке](#))

# Стандартная библиотека

Полная документация

Версия на русском (*вероятно неполная*)

# Модули общего назначения

- ❖ calendar - работа с календарем.
- ❖ datetime - работа с датой и временем (чтение, парсинг, операции).
- ❖ difflib - сравнение последовательностей.
- ❖ hashlib - модуль с набором хеш функций: SHA1, SHA224, SHA256, SHA384, SHA512, MD5...
- ❖ decimal - функции для быстрого преобразования чисел с плавающей точкой.
- ❖ math - функции для работы с математикой.
- ❖ random - генератор псевдо-случайных чисел.
- ❖ re – работа с регулярными выражениями синтаксиса perl.
- ❖ string - функции для работы со строками.

# Модули для работы с различными форматами

- ❖ html – модуль для разбора html страниц.
- ❖ json - модуль для работы с форматом json.
- ❖ xml - модуль для работы с форматом xml.
- ❖ csv - модуль для работы с форматом csv.
- ❖ gzip, zlib - модули для работы со сжатыми данными.
- ❖ email - модуль для разбора структуры email сообщений, проверки email и т.д.
- ❖ configparser - чтение содержимого конфигурационных файлов формата ini.

# Модули Ввода/вывода и взаимодействия с ОС

- ❖ io - основные функции для работы с потоками ввода/вывода (текст, бинарные...)
- ❖ os - взаимодействие с операционной системой.
- ❖ pathlib - работа с путями в файловой системе.
- ❖ threading – модуль для работы с многопоточностью.

# Модули для реализации интерфейса и логирования

- ❖ curses - графический интерфейс в терминале.
- ❖ tkinter - стандартный кроссплатформенный графический интерфейс.
- ❖ logging – модуль для логирования.
- ❖ gettext – модуль для добавления локализации.

# Модули для работы с сетью

- ❖ http - работа с интернет ресурсами по протоколу HTTP.
- ❖ ssl - работа с ssl сертификатами, используется для получения html страниц по протоколу https.
- ❖ socket - работа с сокетами напрямую.
- ❖ urllib - простая работа с URL.

Сторонние модули

# Пакеты для отображения данных

- ❖ Matplotlib – пакет для визуализации данных с богатым функционалом построения графиков
- ❖ Seaborn – расширение для Matplotlib, направленное на то, чтобы сделать графики Matplotlib привлекательнее и упростить создание сложных визуализаций.
- ❖ Plotly - интерактивные графики, позволяющие исследовать взаимоотношения переменных. Продвинутый функционал по построению трехмерных графиков
  
- ❖ Scikit-Learn
- ❖ Keras

# Пакеты для DS

- ❖ NumPy – быстрые вычисления, математические структуры данных (многомерные массивы), множество готовых математических функций
- ❖ SciPy – основан на NumPy, набор подпакетов, в которых реализованы различные вычислительные механизмы (быстрое преобразование Фурье, решение дифференциальных уравнений, механизмы линейной алгебры...)
- ❖ Pandas – использует NumPy и SciPy, предназначен для анализа данных в том числе больших объемов.
- ❖ StatsModels – использует NumPy и SciPy, также используется для анализа данных и построения статистических моделей.

# ML

- ❖ TensorFlow – библиотека для машинного обучения (не только для python)
- ❖ Keras – надстройка над TensorFlow
- ❖ Scikit-Learn

Как хранить кодовую  
базу

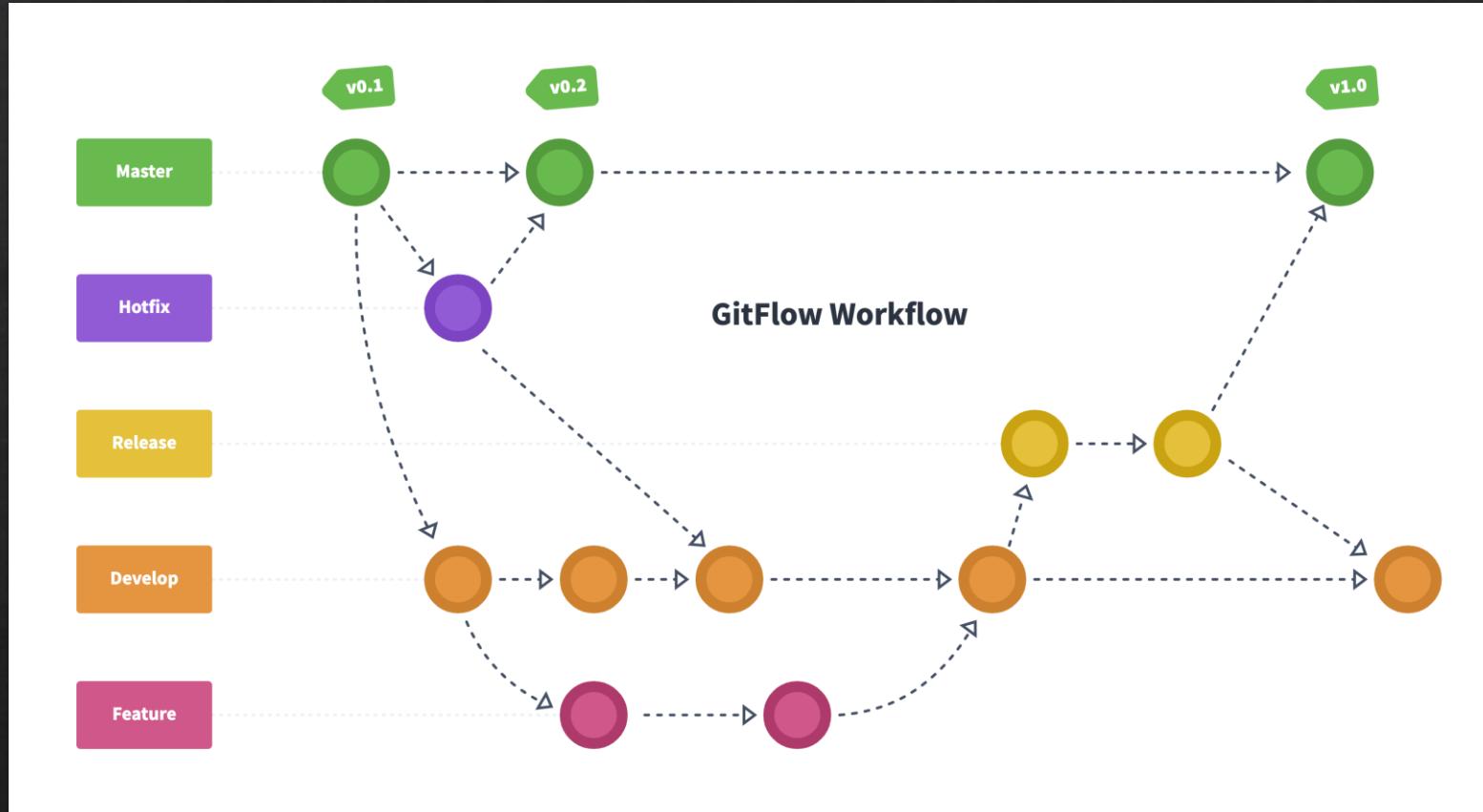
# Хранение и версионирование кода

- ❖ Проблемы:
  - ❖ Много разработчиков
  - ❖ Много версий
  - ❖ Потеря данных
- ❖ Решение: система контроля версий (git)
- ❖ Разработчики пользуются ПО (github, gitlab) предоставляющим git репозиторий и набор дополнительных инструментов вокруг

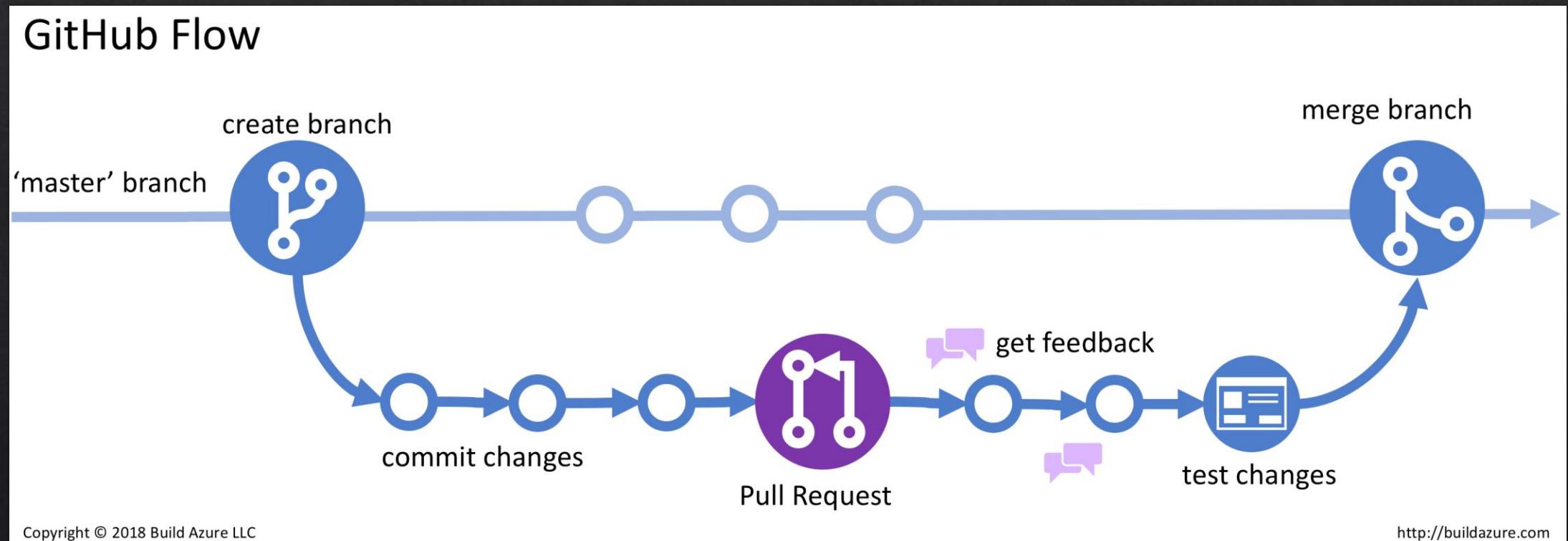
# Основы git

- ❖ Единица хранения – коммит (фиксация) – набор изменений относительно предыдущего коммита.
- ❖ Коммиты объединяются в ветки, позволяющие вести параллельную разработку и реже заниматься разрешением конфликтов
- ❖ Есть удаленный и локальный репозитории

# GIT FLOW



# Github flow



# GIT commands

- ❖ git clone
- ❖ git checkout (-b)
- ❖ git status
- ❖ git add / rm
- ❖ git commit
- ❖ git push
- ❖ git merge

Демонстрация - Что делать в ЛР