Классы

Методы и свойства

Магические методы

Наследование

Исключения

Классы

`type(obj)` тип объекта

`isinstance(obj, cls)` принадлежность объекта классу

`class_name.__dict__` атрибуты класса

`obj_name.__class__.__dict__`

`obj_name.__dict__` атрибуты объекта (self)

`getattr(class_name, atr_name, default)` значение атрибута

`setattr(obj, atr_name, value)` установить значение атрибута

`hesattr(obj, atr_name)` проверяет есть ли атрибут

`delattr(class_name, atr_name)` удалить атрибут класса

`del class_name.atr_name` удалить атрибут класса

пространство имен класса (глобально)

пространство имен экземпляра (локально для каждого экземпляра)

поиск сначала в экземпляре потом в классе

`@staticmethod` декоратор метода класса вызывается как в пространстве имен класса так и экземпляра класса

инициализация сначала `__new__` затем `__init__`

`dir()`

модуль `accessify` приватный и защищённый

var = property(fget=None, fset=None, fdel=None, doc=None) создания свойства

```
class Cl:

    _x = 0

    def __init__(self):
        print('__init__')
        self._x = Cl._x
```

```
    def set_x(self, v):
        print(f'set x {v}')
        self._x = v

    def get_x(self):
        print('get x')
        return self._x

    x = property(fget=get_x, fset=set_x, fdel=None, doc=None)
```

или

```
xy = property()
xy = xy.getter(get_x)
xy = xy.setter(set_x)
xy = xy.deleter(None)
```

```
def xy(self):
    print('get xy')
    return self._xy
# декарируем метод превращая его в свойства
xy = property(xy)
```

```
# декарируем метод превращая его в свойства
@property
def xy(self):
    print('get xy')
    return self._xy

@xy.setter
def xy(self, v):
    print('set xy')
    if not isinstance(v, (int, float)):
        raise ValueError('not a number')
    self._xy = v

@xy.deleter
def xy(self):
    print('del xy')
    del self._xy
```

`from string import digits`

Магические методы

магические методы

double underscore

dunder метод

`__str__` `__repr__` текстовое отображение в системе

представление объекта, str print

`__len__` `__abs__` длина абсолютное значение

`__add__ __radd__ __mul__ __sub__ __truediv__`

```
class Bank:
    def __init__(self, balance):
        self.balance = balance

    def __add__(self, var):
        if isinstance(var, (int, float)):
            print(f'add {var}')
            self.balans += var
            return self.balance
        if isinstance(var, Bank):
            return self.balance + var.balance
        raise NotImplemented
```

`__eq__ ==` `__ne__ !=` `__lt__ <` `__le__ <=` `__gt__ >` `__ge__ >=`

`__hesh__` значение ключей словаря (если поддерживает класс)

`__eq__` переопределение ломает определение `__hesh__` по умолчанию

```
    def __hash__(self):
        return hash((self.x, self.y))


d = {}
cl = Cls()
d[cl] = 100
```

`__call__` вызов экземпляра класса как функцию

использование класса как декоратора

```
def __call__(self, *args, **kwargs):
    pass
```

```
from time import perf_counter
from math import sin

class TimerCall:
    def __init__(self, func):
        self.func = func

    def __call__(self, *args, **kwargs):
        print(f'call func {self.func.__name__}')
        start = perf_counter()
        func_rez = self.func(*args, **kwargs)
        stop = perf_counter()
        print(f'time run func {stop-start}')
        return func_rez


def fac(n):
    if n == 1:
        return 1
    else:
```

```
        return (n * fac(n-1))


cl = TimerCall(fac)
r = cl(10)
print(r)


@TimerCall
def temp_func():
    print('temp func start')
    s = 0
    for i in range(100000):
        s += sin(i)
    print('temp func stop')

temp_func()
```

`__setitem__ getitem__ __delitem__`

`def __setitem__(self, key, val): pass`

`c[key]=val`

`def __getitem__(self, key)`: pass

`x=c[key]`

`def __delitem__(self, key): pass`

`del c[key]`

`raise IndexError('Error massage')`

`val_list.extend(list, set, str, iter )`

`__iter__ __next__ yald`

`next()`

`iter()`

`StopIeration`

```
class Vector:

    def __init__(self, v):
        self.vv = v

    def __iter__(self):
        print('__iter__')
        self.v = self.vv
        # return iter('1234567890')
        return self

    def __next__(self):
        self.v -= 1
        if self.v < 0:
            self.v = self.vv
            raise StopIteration
```

```
        return self.v

vv = Vector(10)

for i in vv:
    print(i)
```

Наследование

`issubclass() isintance`

overriding переопределение метода и атрибута

полиморфизм

extending расширение описываются методы и атрибуты без реализации
реализация в классах наследников

`hasattr(self, 'func')`

делегирование вызов функции родителя через `super().func()`

множественное наследие

`__mro__` порядок классов поиска методов при множественном наследовании

Исключения

в момент выполнения

```
try:
except:
```

Base Exception -> Exception SystemExit GeneratorExit KeyboardInterrupt

Attribute Error Arithmetic Error EOF Error Name Error Lookup Error OS Error Type Error
Type Error Value Error

`raise ValueError('text error')` вызов исключений

```
try:
except ValueError:
except ZeroDivisionError:
except NameError:
except:
except ( , , , ):
except KeyError as ke:
else: # только если нет исключений
finally: # в любом случае выполниться
```

```
except (KeyError, IndexError) as er:
    print('except error')
    raise # генерируем исключение
    # или
    raisse TypeError('error mes') from None # не выводить в консоль преддыдущие
исключения
```

вложение исключения

пользовательские исключения

наследование от Exception