Python

Group session #2

f-string

Makes string interpolation simpler!

```
print("Hello, My name is " + name + " and I'm " + str(age) + " years old.")

print("Hello, My name is", name, "and I'm", age, "years old.")

print(f"Hello, My name is {name} and I'm {age} years old.")
```

Check `type()` of variable

`isinstance(a, b)` checks if a is of type b.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age


name = "John"
age = 36

person = Person(name, age)

print(type(name))
print(type(age))
print(type(person))
```

✓ 0.1s

```
<class 'str'>
<class 'int'>
<class '__main__.Person'>
```

ALWAYS add docstring to a function!!!

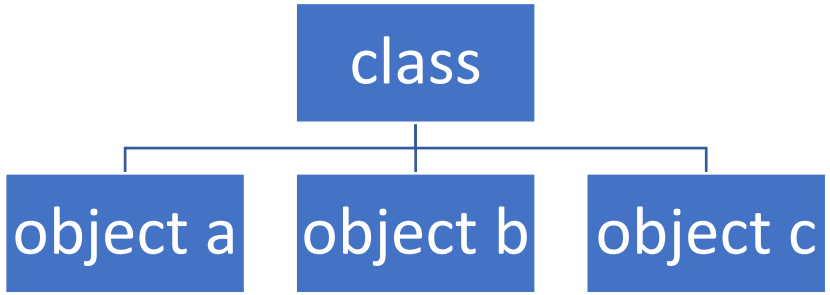
```
def __str__(self):
    """Returns a nicely printable string representation of the array.

    Returns:
        str: A string representation of the array.

    """
    pass
```

Object-oriented programming (OOP)

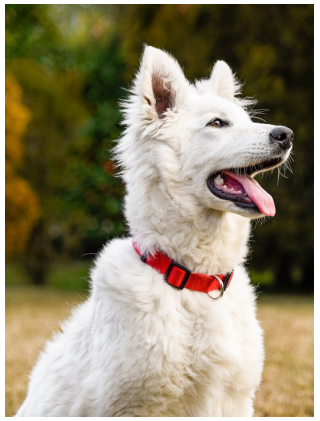
- Idea: Stores data and code in “objects”.
- Usually class-based.
- Classes and objects were introduced by Simula 67 by Ole-Johan Dahl and Kristen Nygaard.



simula

<class '__main__.Dog'>





Different “Dog objects”

```python
class Dog:

    def __init__(self, name, sex):
        """Initialize a dog object.

        Args:
            name (str): The dogs name.
            sex (str): The dogs sex.

        Raises:
            TypeError: If "shape" or "values" are of the wrong type.
        """

        self.name = name
        self.sex = sex

        #check if name is type str
        if not isinstance(self.name, str):
            raise TypeError(f"name must be a string, and not type {type(self.name)}")

        #check if sex is type str
        if not isinstance(self.sex, str):
            raise TypeError(f"name must be a string, and not type {type(self.sex)}")
```

Dunder Methods

- Dunder means “Double Under (Underscores)”
- Also called “magic methods”

```
def __str__(self):
    """Returns a nicely printable string representation of the dog.

    Returns:
        str: A string representation of the dog.
    """
    return f"{self.name} is a {self.sex} dog."
```

Without `__str()__`:

```
Daisy = Dog("Daisy", "female")
print(Daisy)
```

✓ 0.9s

```
<dog.Dog object at 0x7fc1e601a0a0>
```

With `__str()__`:

```
Daisy = Dog("Daisy", "female")
print(Daisy)
```

✓ 0.7s

```
Daisy is a female dog.
```

`__add__` vs `__radd__`

We have: `x + y`

Python first tries to call the left object's `__add__()` method `x.__add__(y)`. But this may fail for two reasons:

1. The method `x.__add__()` is not implemented in the first place,
2. The method `x.__add__()` is implemented but returns a `NotImplemented value` indicating that the data types are incompatible.

If this fails, Python tries to fix it by calling the `y.__radd__()` for *reverse addition* on the right operand `y`.

Unit tests

1. Identify a *unit* in your program that should have a well-defined behavior given a certain input. A unit can be a:
    1. function
    2. module
    3. entire program
2. Write a test function that calls this input and checks that the output/behavior is as expected.
3. The more, the better! Preferably on several levels (function/module/program).
4. Use a test framework like py.test and checks like `assert`

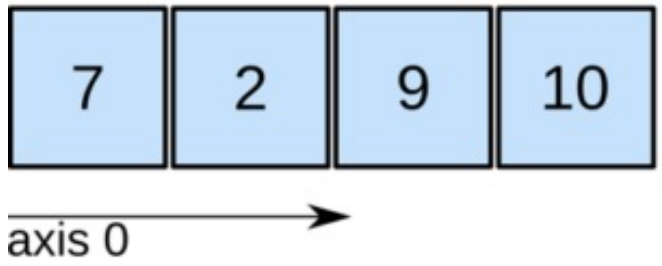
Unit tests

```python
def test_dog():
    """ Test for dog class"""
    dog = Dog("Pluto", "male")

    assert dog.name == "Pluto"
    assert isinstance(dog.sex, str)
    assert dog.__str__() == "Pluto is a male dog."
```

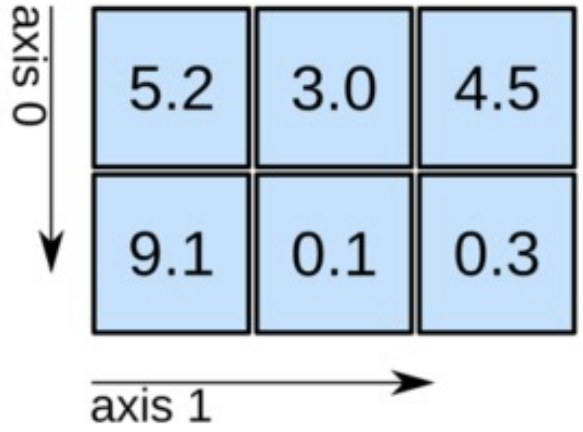
Arrays

1D array



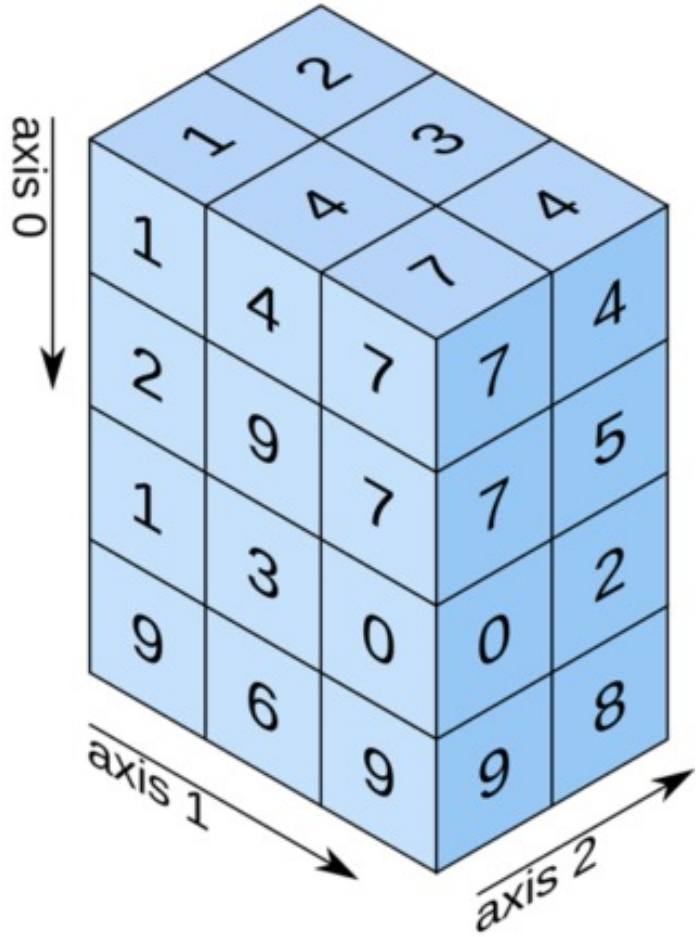
shape: (4,)

2D array



shape: (2, 3)

3D array



shape: (4, 3, 2)

https://www.pythontutorial.net/python-numpy/what-is-numpy/