Complex Systems & Agent-Based Modelling

Lecture 6

Agostino Merico

08 February 2018

What is a complex system ?

http://www.youtube.com/v/3w90X92pDSs?rel=0

Complex Systems

Complex Systems is a field of science studying how individual components of a system give rise to collective behaviours and how the system interacts with its environment.

Complex Systems

Complex Systems is a field of science studying how individual components of a system give rise to collective behaviours and how the system interacts with its environment.

Social systems formed out of people, the brain formed out of neurons, molecules formed out of atoms, the weather formed out of air flows are all examples of complex systems.

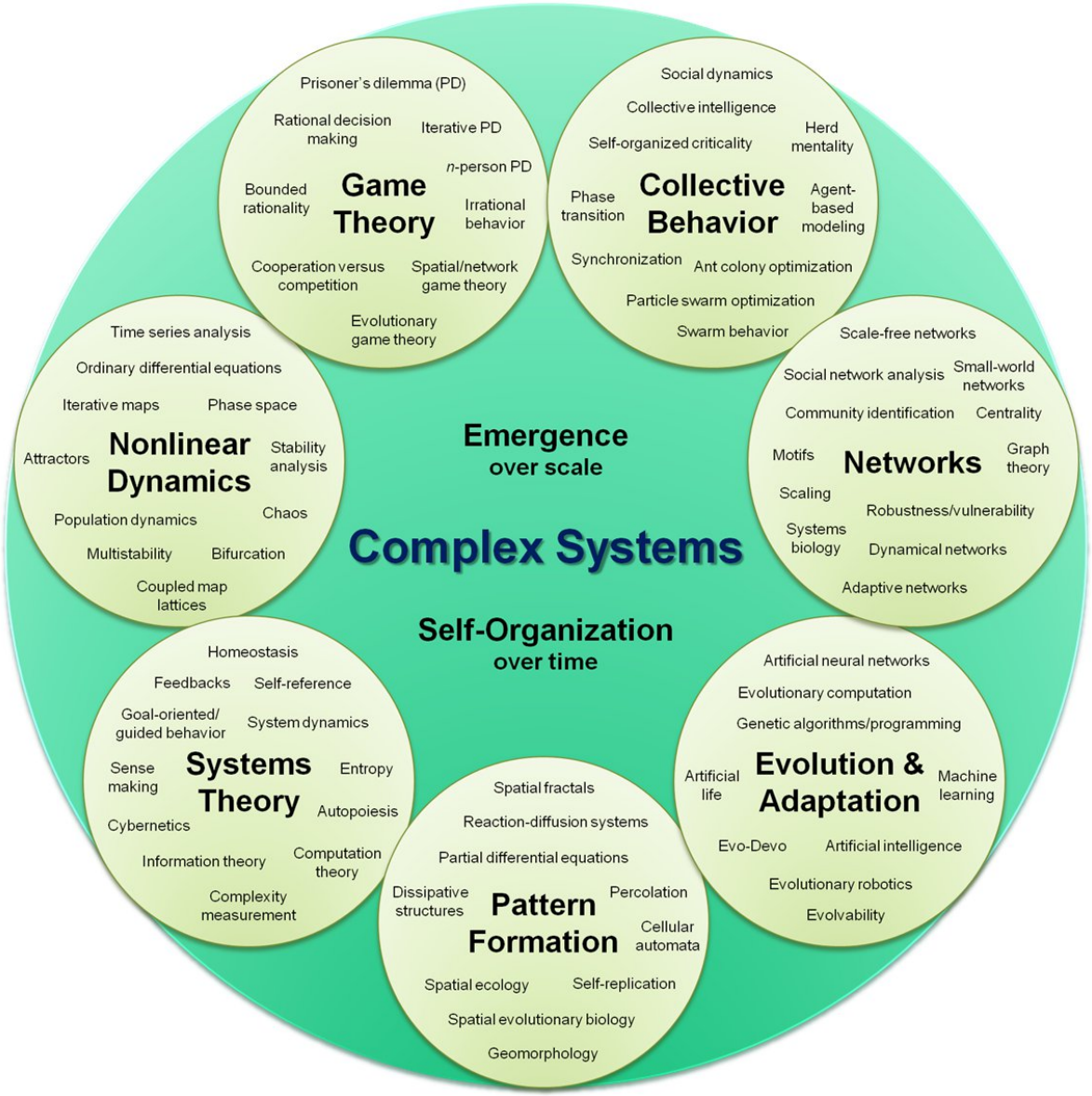
Complex Systems

Complex Systems is a field of science studying how individual components of a system give rise to collective behaviours and how the system interacts with its environment.

Social systems formed out of people, the brain formed out of neurons, molecules formed out of atoms, the weather formed out of air flows are all examples of complex systems.

The field of complex systems cuts across all traditional disciplines of science, as well as engineering, management, and medicine.

Organizational map of Complex Systems



What are agent-based models ?

Agent-Based Models (ABMs) are arguably the most generalized tool for modelling and simulating **complex systems**, they include both Cellular Automata and Dynamical Networks models as special cases.

Applications of agent-based models

ABMs are widely used in a variety of disciplines to simulate dynamical behaviours of systems made of a large number of entities, such as

Applications of agent-based models

ABMs are widely used in a variety of disciplines to simulate dynamical behaviours of systems made of a large number of entities, such as

- Trader's behaviour in a market (economics);

Applications of agent-based models

ABMs are widely used in a variety of disciplines to simulate dynamical behaviours of systems made of a large number of entities, such as

- Trader's behaviour in a market (economics);
- Migration of people (social sciences);

Applications of agent-based models

ABMs are widely used in a variety of disciplines to simulate dynamical behaviours of systems made of a large number of entities, such as

- Trader's behaviour in a market (economics);
- Migration of people (social sciences);
- Interaction among employees and their performance improvement (organizational science);

Applications of agent-based models

ABMs are widely used in a variety of disciplines to simulate dynamical behaviours of systems made of a large number of entities, such as

- Trader's behaviour in a market (economics);
- Migration of people (social sciences);
- Interaction among employees and their performance improvement (organizational science);
- Flocking / schooling behaviour of birds / fish (behavioural ecology);

Applications of agent-based models

ABMs are widely used in a variety of disciplines to simulate dynamical behaviours of systems made of a large number of entities, such as

- Trader's behaviour in a market (economics);
- Migration of people (social sciences);
- Interaction among employees and their performance improvement (organizational science);
- Flocking / schooling behaviour of birds / fish (behavioural ecology);
- Cell growth and morphogenesis (developmental biology):

Applications of agent-based models

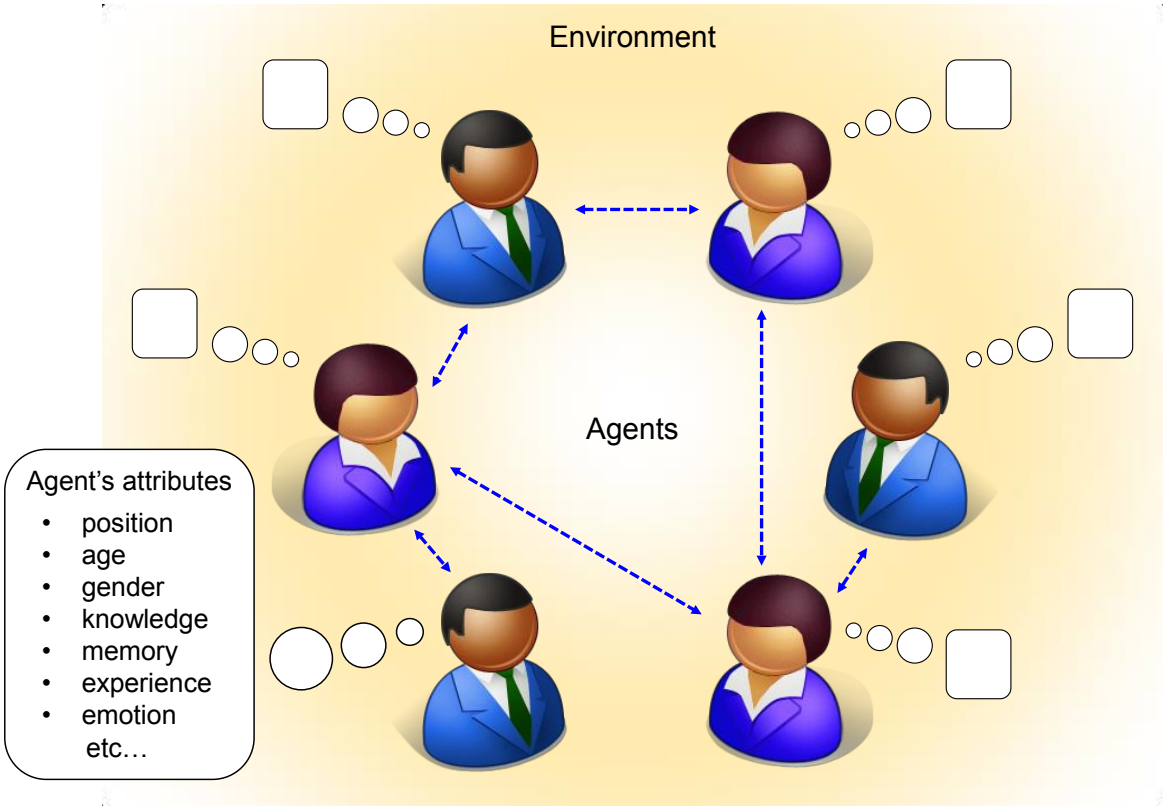
ABMs are widely used in a variety of disciplines to simulate dynamical behaviours of systems made of a large number of entities, such as

- Trader's behaviour in a market (economics);
- Migration of people (social sciences);
- Interaction among employees and their performance improvement (organizational science);
- Flocking / schooling behaviour of birds / fish (behavioural ecology);
- Cell growth and morphogenesis (developmental biology):
- Collective behaviour of granular materials (physics).

Definition of agent-based models

Definition: Agent-based models are computational simulation models that involve many discrete agents.

Illustration of an agent-based model



Schametic illustration of what an agent-based model looks like.

Main features of ABMs

Definition: Agent-based models are computational simulation models that involve many discrete agents.

Main features of ABMs

Definition: Agent-based models are computational simulation models that involve many discrete agents.

Computational. ABMs are usually implemented as simulation models in a computer, where each agents behavioural rules are described in an *algorithmic* fashion rather than a purely mathematical way.

Main features of ABMs

Definition: Agent-based models are computational simulation models that involve many discrete agents.

Computational. ABMs are usually implemented as simulation models in a computer, where each agents behavioural rules are described in an *algorithmic* fashion rather than a purely mathematical way.

This allows modellers to implement complex *internal properties* of agents and their non-trivial behavioural rules.

Main features of ABMs

Definition: Agent-based models are computational simulation models that involve many discrete agents.

Computational. ABMs are usually implemented as simulation models in a computer, where each agents behavioural rules are described in an *algorithmic* fashion rather than a purely mathematical way.

This allows modellers to implement complex *internal properties* of agents and their non-trivial behavioural rules.

Such representations of complex individual traits are highly valued especially in social, organizational and management sciences, where the modellers need to capture complex, realistic behaviours of human individuals. This is why ABMs are particularly popular in those research areas.

Main features of ABMs

Definition: Agent-based models are computational simulation models that involve many discrete agents.

Main features of ABMs

Definition: Agent-based models are computational simulation models that involve many discrete agents.

Many. Although it is technically possible to create an ABM made of just a few agents, there would be little need for such a model.

Main features of ABMs

Definition: Agent-based models are computational simulation models that involve many discrete agents.

Many. Although it is technically possible to create an ABM made of just a few agents, there would be little need for such a model.

The typical context in which an ABM is needed is when researchers want to study the *collective behaviour* of a large number of agents, otherwise it would be sufficient to use a more conventional equation-based model with a small number of variables.

Main features of ABMs

Definition: Agent-based models are computational simulation models that involve many discrete agents.

Main features of ABMs

Definition: Agent-based models are computational simulation models that involve many discrete agents.

Discrete. While there are some ambiguities about how to rigorously define an agent, what is commonly accepted is that an agent should be a discrete individual entity.

Main features of ABMs

Definition: Agent-based models are computational simulation models that involve many discrete agents.

Discrete. While there are some ambiguities about how to rigorously define an agent, what is commonly accepted is that an agent should be a discrete individual entity.

Cellular Automata and Network Models are made of discrete components, so they qualify as special cases of ABMs.

Main features of ABMs

Definition: Agent-based models are computational simulation models that involve many discrete agents.

Discrete. While there are some ambiguities about how to rigorously define an agent, what is commonly accepted is that an agent should be a discrete individual entity.

Cellular Automata and Network Models are made of discrete components, so they qualify as special cases of ABMs.

Time-continuous, differential equation models adopt continuous spatial functions as a representation of the system's state, so they are not considered ABMs.

Typical properties of agents

There are certain properties that are generally assumed for agents, these are:

Typical properties of agents

There are certain properties that are generally assumed for agents, these are:

- Agents are discrete entities.

Typical properties of agents

There are certain properties that are generally assumed for agents, these are:

- Agents are discrete entities.
- Agents may have internal states.

Typical properties of agents

There are certain properties that are generally assumed for agents, these are:

- Agents are discrete entities.
- Agents may have internal states.
- Agents may be spatially localised.

Typical properties of agents

There are certain properties that are generally assumed for agents, these are:

- Agents are discrete entities.
- Agents may have internal states.
- Agents may be spatially localised.
- Agents may perceive and interact with the environment.

Typical properties of agents

There are certain properties that are generally assumed for agents, these are:

- Agents are discrete entities.
- Agents may have internal states.
- Agents may be spatially localised.
- Agents may perceive and interact with the environment.
- Agents may behave based on predefined rules.

Typical properties of agents

There are certain properties that are generally assumed for agents, these are:

- Agents are discrete entities.
- Agents may have internal states.
- Agents may be spatially localised.
- Agents may perceive and interact with the environment.
- Agents may behave based on predefined rules.
- Agents may be able to learn and adapt.

Typical properties of agents

There are certain properties that are generally assumed for agents, these are:

- Agents are discrete entities.
- Agents may have internal states.
- Agents may be spatially localised.
- Agents may perceive and interact with the environment.
- Agents may behave based on predefined rules.
- Agents may be able to learn and adapt.
- Agents may interact with other agents.

Typical properties of agents

There are certain properties that are generally assumed for agents, these are:

- Agents are discrete entities.
- Agents may have internal states.
- Agents may be spatially localised.
- Agents may perceive and interact with the environment.
- Agents may behave based on predefined rules.
- Agents may be able to learn and adapt.
- Agents may interact with other agents.
- ABMs often lack central supervisors / controllers.

Typical properties of agents

There are certain properties that are generally assumed for agents, these are:

- Agents are discrete entities.
- Agents may have internal states.
- Agents may be spatially localised.
- Agents may perceive and interact with the environment.
- Agents may behave based on predefined rules.
- Agents may be able to learn and adapt.
- Agents may interact with other agents.
- ABMs often lack central supervisors / controllers.
- ABMs may produce non-trivial "collective behaviour" as a whole.

Typical properties of agents

Note that these are not strict requirements for ABMs (perhaps except for the first one).

Typical properties of agents

Note that these are not strict requirements for ABMs (perhaps except for the first one).

Some ABMs do not have internal states of agents; some do not have space or environment; and some *do* have central controllers as special kinds of agents.

Typical properties of agents

Note that these are not strict requirements for ABMs (perhaps except for the first one).

Some ABMs do not have internal states of agents; some do not have space or environment; and some *do* have central controllers as special kinds of agents.

Therefore, which model properties should be incorporated into an ABM is really up to the objective of your model.

General aspects to consider for developing ABMs

General aspects to consider for developing ABMs

Implementing an ABM is usually much more coding-intense than implementing other simpler models, partly because the ABM framework is open-ended.

General aspects to consider for developing ABMs

Implementing an ABM is usually much more coding-intense than implementing other simpler models, partly because the ABM framework is open-ended.

This naturally increases the amount of coding you will need to do. And, the more you code, the more likely unexpected bugs will sneak into the code.

General aspects to consider for developing ABMs

Implementing an ABM is usually much more coding-intense than implementing other simpler models, partly because the ABM framework is open-ended.

This naturally increases the amount of coding you will need to do. And, the more you code, the more likely unexpected bugs will sneak into the code.

It is thus very important to keep the code simple and organized, and to use the best practices in computer programming, e.g. modularizing sections, adding plenty of comments, implementing systematic tests, etc.

General aspects to consider for developing ABMs

Implementing an ABM is usually much more coding-intense than implementing other simpler models, partly because the ABM framework is open-ended.

This naturally increases the amount of coding you will need to do. And, the more you code, the more likely unexpected bugs will sneak into the code.

It is thus very important to keep the code simple and organized, and to use the best practices in computer programming, e.g. modularizing sections, adding plenty of comments, implementing systematic tests, etc.

Since ABMs are so open-ended and flexible, modellers are tempted to add more and more complex settings and assumptions into their ABMs.

General aspects to consider for developing ABMs

Implementing an ABM is usually much more coding-intense than implementing other simpler models, partly because the ABM framework is open-ended.

This naturally increases the amount of coding you will need to do. And, the more you code, the more likely unexpected bugs will sneak into the code.

It is thus very important to keep the code simple and organized, and to use the best practices in computer programming, e.g. modularizing sections, adding plenty of comments, implementing systematic tests, etc.

Since ABMs are so open-ended and flexible, modellers are tempted to add more and more complex settings and assumptions into their ABMs.

But beware – the increased model complexity increases the difficulty of analysing and justifying the model and its results.

General aspects to consider for developing ABMs

Implementing an ABM is usually much more coding-intense than implementing other simpler models, partly because the ABM framework is open-ended.

This naturally increases the amount of coding you will need to do. And, the more you code, the more likely unexpected bugs will sneak into the code.

It is thus very important to keep the code simple and organized, and to use the best practices in computer programming, e.g. modularizing sections, adding plenty of comments, implementing systematic tests, etc.

Since ABMs are so open-ended and flexible, modellers are tempted to add more and more complex settings and assumptions into their ABMs.

But beware – the increased model complexity increases the difficulty of analysing and justifying the model and its results.

If we want to derive useful, reliable conclusions from the model, we should resist the temptation to add unecessary complexity.

Building an Agent-Based Model

The specific aspects to consider when designing an agent-based model are:

Building an Agent-Based Model

The specific aspects to consider when designing an agent-based model are:

1. Specific problem to be solved by the ABM.

Building an Agent-Based Model

The specific aspects to consider when designing an agent-based model are:

1. Specific problem to be solved by the ABM.
2. Design of agents and their static / dynamic attributes.

Building an Agent-Based Model

The specific aspects to consider when designing an agent-based model are:

1. Specific problem to be solved by the ABM.
2. Design of agents and their static / dynamic attributes.
3. Design of an environment and the way agents interact with it.

Building an Agent-Based Model

The specific aspects to consider when designing an agent-based model are:

1. Specific problem to be solved by the ABM.
2. Design of agents and their static / dynamic attributes.
3. Design of an environment and the way agents interact with it.
4. Design of agents' behaviours.

Building an Agent-Based Model

The specific aspects to consider when designing an agent-based model are:

1. Specific problem to be solved by the ABM.
2. Design of agents and their static / dynamic attributes.
3. Design of an environment and the way agents interact with it.
4. Design of agents' behaviours.
5. Design of agents' mutual interactions.

Building an Agent-Based Model

The specific aspects to consider when designing an agent-based model are:

1. Specific problem to be solved by the ABM.
2. Design of agents and their static / dynamic attributes.
3. Design of an environment and the way agents interact with it.
4. Design of agents' behaviours.
5. Design of agents' mutual interactions.
6. Availability of data.

Building an Agent-Based Model

The specific aspects to consider when designing an agent-based model are:

1. Specific problem to be solved by the ABM.
2. Design of agents and their static / dynamic attributes.
3. Design of an environment and the way agents interact with it.
4. Design of agents' behaviours.
5. Design of agents' mutual interactions.
6. Availability of data.
7. Method of model validation.

Building an Agent-Based Model

The specific aspects to consider when designing an agent-based model are:

1. Specific problem to be solved by the ABM.
2. Design of agents and their static / dynamic attributes.
3. Design of an environment and the way agents interact with it.
4. Design of agents' behaviours.
5. Design of agents' mutual interactions.
6. Availability of data.
7. Method of model validation.

Points 1, 6, and 7 are about fundamental scientific methodologies.

Building an Agent-Based Model

The specific aspects to consider when designing an agent-based model are:

1. Specific problem to be solved by the ABM.
2. Design of agents and their static / dynamic attributes.
3. Design of an environment and the way agents interact with it.
4. Design of agents' behaviours.
5. Design of agents' mutual interactions.
6. Availability of data.
7. Method of model validation.

Points 1, 6, and 7 are about fundamental scientific methodologies.

Typically, IBM are built using assumptions that are derived from observations to produce previously unknown collective behaviours with the simulations.

Coding an Agent-Based Model

Points 2, 3, 4, and 5 in the previous list are more focused on the technical aspects of modelling. They can be translated into the following design tasks in actual python coding:

Coding an Agent-Based Model

Points 2, 3, 4, and 5 in the previous list are more focused on the technical aspects of modelling. They can be translated into the following design tasks in actual python coding:

1. Design the data structure to store the attributes of the agents.

Coding an Agent-Based Model

Points 2, 3, 4, and 5 in the previous list are more focused on the technical aspects of modelling. They can be translated into the following design tasks in actual python coding:

1. Design the data structure to store the attributes of the agents.
2. Design the data structure to store the states of the environment.

Coding an Agent-Based Model

Points 2, 3, 4, and 5 in the previous list are more focused on the technical aspects of modelling. They can be translated into the following design tasks in actual python coding:

1. Design the data structure to store the attributes of the agents.
2. Design the data structure to store the states of the environment.
3. Describe the rules for how the environment behaves on its own.

Coding an Agent-Based Model

Points 2, 3, 4, and 5 in the previous list are more focused on the technical aspects of modelling. They can be translated into the following design tasks in actual python coding:

1. Design the data structure to store the attributes of the agents.
2. Design the data structure to store the states of the environment.
3. Describe the rules for how the environment behaves on its own.
4. Describe the rules for how agents interact with the environment.

Coding an Agent-Based Model

Points 2, 3, 4, and 5 in the previous list are more focused on the technical aspects of modelling. They can be translated into the following design tasks in actual python coding:

1. Design the data structure to store the attributes of the agents.
2. Design the data structure to store the states of the environment.
3. Describe the rules for how the environment behaves on its own.
4. Describe the rules for how agents interact with the environment.
5. Describe the rules for how agents behave on their own.

Coding an Agent-Based Model

Points 2, 3, 4, and 5 in the previous list are more focused on the technical aspects of modelling. They can be translated into the following design tasks in actual python coding:

1. Design the data structure to store the attributes of the agents.
2. Design the data structure to store the states of the environment.
3. Describe the rules for how the environment behaves on its own.
4. Describe the rules for how agents interact with the environment.
5. Describe the rules for how agents behave on their own.
6. Describe the rules for how agents interact with each other.

Representation of agents in python

It is often convenient and customary to define both agents' attributes and behaviours using a `class`, which is a fundamental element in object-oriented programming. More on pyhton classes here.

Representation of agents in python

It is often convenient and customary to define both agents' attributes and behaviours using a `class`, which is a fundamental element in object-oriented programming. More on pyhton classes here.

In this course, we will use the python's dynamic class as a pure data structure to store agents' attributes in a concise manner.

Representation of agents in python

It is often convenient and customary to define both agents' attributes and behaviours using a `class`, which is a fundamental element in object-oriented programming. More on pyhton classes here.

In this course, we will use the python's dynamic class as a pure data structure to store agents' attributes in a concise manner.

For example, we can define an empty class called `agent` as follows:

```
class agent:
    pass
```

Representation of agents in python

It is often convenient and customary to define both agents' attributes and behaviours using a `class`, which is a fundamental element in object-oriented programming. More on pyhton classes here.

In this course, we will use the python's dynamic class as a pure data structure to store agents' attributes in a concise manner.

For example, we can define an empty class called `agent` as follows:

```
class agent:
    pass
```

Once this `agent` class is defined, we can create a new empty `agent`: object as follows

Representation of agents in python

It is often convenient and customary to define both agents' attributes and behaviours using a `class`, which is a fundamental element in object-oriented programming. More on pyhton classes here.

In this course, we will use the python's dynamic class as a pure data structure to store agents' attributes in a concise manner.

For example, we can define an empty class called `agent` as follows:

```
class agent:
    pass
```

Once this `agent` class is defined, we can create a new empty `agent`: object as follows

```
>>> ag1 = agent()
```

Representation of agents in python

Then we can dynamically add various attributes to the object ag1:

Representation of agents in python

Then we can dynamically add various attributes to the object ag1:

```
>>> ag1.x = 2
>>> ag1.y = 8
>>> ag1.name = 'John'
>>> ag1.age = 21
```

Representation of agents in python

Then we can dynamically add various attributes to the object ag1:

```
>>> ag1.x = 2
>>> ag1.y = 8
>>> ag1.name = 'John'
>>> ag1.age = 21

>>> ag1.x
2
```

Representation of agents in python

Then we can dynamically add various attributes to the object ag1:

```
>>> ag1.x = 2
>>> ag1.y = 8
>>> ag1.name = 'John'
>>> ag1.age = 21

>>> ag1.x
2

>>> ag1.name
'John'
```

Representation of agents in python

If you want to know what kinds of attributes are available under an object, you can use the `dir` command:

Representation of agents in python

If you want to know what kinds of attributes are available under an object, you can use the `dir` command:

```
>>> dir(ag1)
['__doc__', '__module__', 'age', 'name', 'x', 'y']
```

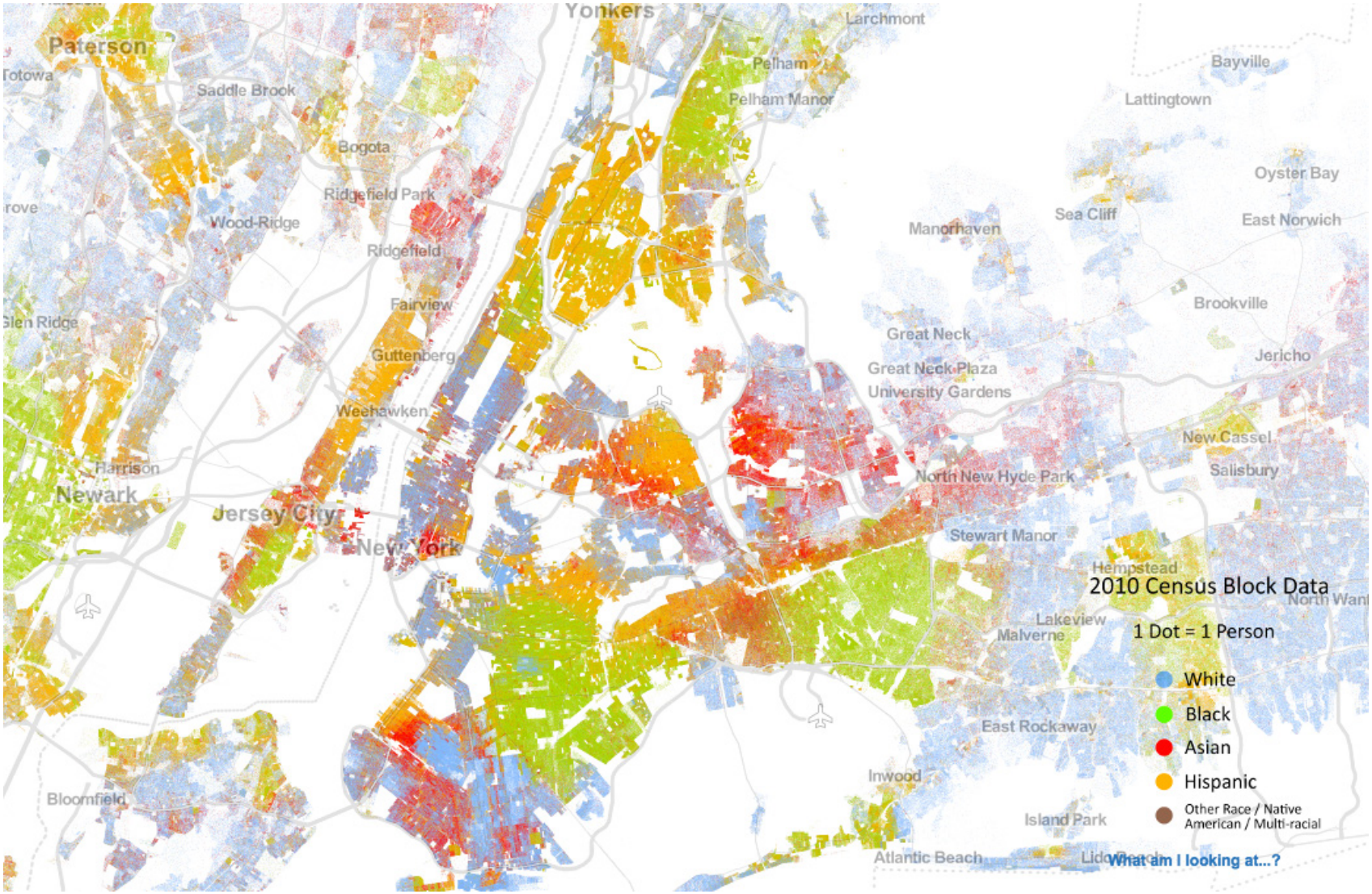
Representation of agents in python

If you want to know what kinds of attributes are available under an object, you can use the `dir` command:

```
>>> dir(ag1)
['__doc__', '__module__', 'age', 'name', 'x', 'y']
```

The first two attributes are python's default attributes that are available for any objects (you can ignore them). Aside from those, we see there are four attributes defined for the object `ag1` .

Building an Agent-Based Model



If you look at the cultural distribution of multi-ethnic cities, you will be astonished by the incredible cultural segregation. For example, the New York City map above represents data from the 2010 US Census color-coded by ethnic origin. You can clearly see the cultural segregation.

Building an Agent-Based Model

Example: Schelling's segregation model

From this map, many would conclude that people are intolerant, and do not want to live next to people that are different from them. A deeper look, however, provides different answers.

Building an Agent-Based Model

Example: Schelling's segregation model

From this map, many would conclude that people are intolerant, and do not want to live next to people that are different from them. A deeper look, however, provides different answers.

In the 70's, Thomas Schelling, the 2005 Economics Nobel Prize winner, was interested in this subject and built an agent-based model called the "Schelling segregation model" for investigating this phenomenon.

Building an Agent-Based Model

Example: Schelling's segregation model

From this map, many would conclude that people are intolerant, and do not want to live next to people that are different from them. A deeper look, however, provides different answers.

In the 70's, Thomas Schelling, the 2005 Economics Nobel Prize winner, was interested in this subject and built an agent-based model called the "Schelling segregation model" for investigating this phenomenon.

With a very simple agent-based model, Schelling could show that what we see at the macro level may not in fact represent what is going down in the micro level.

Building an Agent-Based Model

Example: Schelling's segregation model

Schelling created this ABM in order to provide an explanation for why people with different ethnic backgrounds tend to segregate geographically.

Building an Agent-Based Model

Example: Schelling's segregation model

Schelling created this ABM in order to provide an explanation for why people with different ethnic backgrounds tend to segregate geographically.

The model assumptions used were the following:

Building an Agent-Based Model

Example: Schelling's segregation model

Schelling created this ABM in order to provide an explanation for why people with different ethnic backgrounds tend to segregate geographically.

The model assumptions used were the following:

- Two different types of agents are distributed in a finite 2D space;

Building an Agent-Based Model

Example: Schelling's segregation model

Schelling created this ABM in order to provide an explanation for why people with different ethnic backgrounds tend to segregate geographically.

The model assumptions used were the following:

- Two different types of agents are distributed in a finite 2D space;
- In each iteration, a randomly chosen agent looks around its neighbourhood and, if the fraction of agents of the same type among its neighbours is below a *threshold*, it jumps to another location randomly chosen in the space.

Example: Schelling's segregation model

So, the rule of this model is extremely simple. The main question Schelling addressed was: how high the threshold had to be in order for segregation to occur ?

Example: Schelling's segregation model

So, the rule of this model is extremely simple. The main question Schelling addressed was: how high the threshold had to be in order for segregation to occur ?

It may be expected that segregation would require highly intolerant agents, in which case the critical threshold might be relatively high, say 80 % or so.

Example: Schelling's segregation model

So, the rule of this model is extremely simple. The main question Schelling addressed was: how high the threshold had to be in order for segregation to occur ?

It may be expected that segregation would require highly intolerant agents, in which case the critical threshold might be relatively high, say 80 % or so.

But what Schelling actually showed was that the critical threshold can be much lower than one would expect.

Building an Agent-Based Model

Example: Schelling's segregation model

So, the rule of this model is extremely simple. The main question Schelling addressed was: how high the threshold had to be in order for segregation to occur ?

It may be expected that segregation would require highly intolerant agents, in which case the critical threshold might be relatively high, say 80 % or so.

But what Schelling actually showed was that the critical threshold can be much lower than one would expect.

Segregation can occur even if people are not so intolerant !

Building an Agent-Based Model

Example: Schelling's segregation model

So, the rule of this model is extremely simple. The main question Schelling addressed was: how high the threshold had to be in order for segregation to occur ?

It may be expected that segregation would require highly intolerant agents, in which case the critical threshold might be relatively high, say 80 % or so.

But what Schelling actually showed was that the critical threshold can be much lower than one would expect.

Segregation can occur even if people are not so intolerant !

At the same time, quite against our intuition, a high level of intolerance can actually result in a well mixed society because agents keep moving without reaching a stationary state.

Building an Agent-Based Model

Example: Schelling's segregation model

Let's design the model step by step, going through the six design tasks listed in slide 11.

Building an Agent-Based Model

Example: Schelling's segregation model

Let's design the model step by step, going through the six design tasks listed in slide 11.

Task 1. *Design the data structure to store the attributes of the agents.*
In this model, each agent has a type attribute as well as a position in the 2-D space. The two types can be represented by 0 and 1, and the spatial position can be anywhere within a unit square. Therefore we can generate each agent as follows:

Building an Agent-Based Model

Example: Schelling's segregation model

Let's design the model step by step, going through the six design tasks listed in slide 11.

Task 1. *Design the data structure to store the attributes of the agents.*
In this model, each agent has a type attribute as well as a position in the 2-D space. The two types can be represented by 0 and 1, and the spatial position can be anywhere within a unit square. Therefore we can generate each agent as follows:

```
class agent:
    pass
```

Building an Agent-Based Model

Example: Schelling's segregation model

Let's design the model step by step, going through the six design tasks listed in slide 11.

Task 1. *Design the data structure to store the attributes of the agents.*
In this model, each agent has a type attribute as well as a position in the 2-D space. The two types can be represented by 0 and 1, and the spatial position can be anywhere within a unit square. Therefore we can generate each agent as follows:

```
class agent:
    pass

ag = agent()
ag.type = randint(2)
ag.x = random()
ag.y = random()
```

Building an Agent-Based Model

Example: Schelling's segregation model

To generate a population of agents, we can write something like this:

Building an Agent-Based Model

Example: Schelling's segregation model

To generate a population of agents, we can write something like this:

```
n = 100    # number of agents
```

Building an Agent-Based Model

Example: Schelling's segregation model

To generate a population of agents, we can write something like this:

```
n = 100    # number of agents

class agent:
    pass
```

Building an Agent-Based Model

Example: Schelling's segregation model

To generate a population of agents, we can write something like this:

```
n = 100   # number of agents

class agent:
    pass

def initialize():
    global agents
    agents = []
    for i in xrange(n):
        ag = agent()
        ag.type = randint(2)
        ag.x = random()
        ag.y = random()
        agents.append(ag)
```

Building an Agent-Based Model

Example: Schelling's segregation model

Let's design the model step by step, going through the six design tasks listed in slide 11.

Building an Agent-Based Model

Example: Schelling's segregation model

Let's design the model step by step, going through the six design tasks listed in slide 11.

Task 2. *Design the data structure to store the states of the environment.*
Task 3. *Describe the rules for how the environment behaves on its own.*
Task 4. *Describe the rules for how agents interact with the environment.*
Schelling's model does not have a separate environment that interacts with agents, so we can skip these design tasks.

Building an Agent-Based Model

Example: Schelling's segregation model

Let's design the model step by step, going through the six design tasks listed in slide 11.

Task 2. *Design the data structure to store the states of the environment.*
Task 3. *Describe the rules for how the environment behaves on its own.*
Task 4. *Describe the rules for how agents interact with the environment.*
Schelling's model does not have a separate environment that interacts with agents, so we can skip these design tasks.

Task 5. *Describe the rules for how agents behave on their own.*
We assume that agents do not do anything by themselves, because their actions (movements) are triggered only by interactions with other agents. So we can ignore this design task too.

Building an Agent-Based Model

Example: Schelling's segregation model

Let's design the model step by step, going through the six design tasks listed in slide 11.

Task 2. *Design the data structure to store the states of the environment.*
Task 3. *Describe the rules for how the environment behaves on its own.*
Task 4. *Describe the rules for how agents interact with the environment.*
Schelling's model does not have a separate environment that interacts with agents, so we can skip these design tasks.

Task 5. *Describe the rules for how agents behave on their own.*
We assume that agents do not do anything by themselves, because their actions (movements) are triggered only by interactions with other agents. So we can ignore this design task too.

Task 6. *Describe the rules for how agents interact with each other.*
Finally, there is something we need to implement. The model assumption says that each agent checks who are in its neighbourhood, and if the fraction of the other agents of the same type is less than a threshold, it jumps to another randomly selected location.

Building an Agent-Based Model

Example: Schelling's segregation model

Let's design the model step by step, going through the six design tasks listed in slide 11.

Task 2. *Design the data structure to store the states of the environment.*
Task 3. *Describe the rules for how the environment behaves on its own.*
Task 4. *Describe the rules for how agents interact with the environment.*
Schelling's model does not have a separate environment that interacts with agents, so we can skip these design tasks.

Task 5. *Describe the rules for how agents behave on their own.*
We assume that agents do not do anything by themselves, because their actions (movements) are triggered only by interactions with other agents. So we can ignore this design task too.

Task 6. *Describe the rules for how agents interact with each other.*
Finally, there is something we need to implement. The model assumption says that each agent checks who are in its neighbourhood, and if the fraction of the other agents of the same type is less than a threshold, it jumps to another randomly selected location.

We need to implement a code that allows each agent to find who is nearby.

Building an Agent-Based Model

Example: Schelling's segregation model

There are several computationally efficient algorithms available for neighbour detection, but here we will use the simplest possible method, that is *exhaustive search*.

Building an Agent-Based Model

Example: Schelling's segregation model

There are several computationally efficient algorithms available for neighbour detection, but here we will use the simplest possible method, that is *exhaustive search*.

With this method all the agents are checked, one by one, to see if they are close enough to the focal agent. This is not computationally efficient (computation time increases quadratically as the number of agents increases), but is straightforward and easy to implement.

Building an Agent-Based Model

Example: Schelling's segregation model

The exhaustive search for neighbour detection can be implemented using python's list comprehension, for example:

Building an Agent-Based Model

Example: Schelling's segregation model

The exhaustive search for neighbour detection can be implemented using python's list comprehension, for example:

```
neighbours = [nb for nb in agents
              if (ag.x - nb.x)**2 + (ag.y - nb.y)**2 < r**2 and nb != ag]
```

Building an Agent-Based Model

Example: Schelling's segregation model

The exhaustive search for neighbour detection can be implemented using python's list comprehension, for example:

```
neighbours = [nb for nb in agents
              if (ag.x - nb.x)**2 + (ag.y - nb.y)**2 < r**2 and nb != ag]
```

Here ag is the focal agent whose neighbours, nb, are searched for.

Building an Agent-Based Model

Example: Schelling's segregation model

The exhaustive search for neighbour detection can be implemented using python's list comprehension, for example:

```
neighbours = [nb for nb in agents
              if (ag.x - nb.x)**2 + (ag.y - nb.y)**2 < r**2 and nb != ag]
```

Here `ag` is the focal agent whose neighbours, `nb`, are searched for.

The `if` part in the list comprehension measures the distance squared between `ag` and `nb`, and if it is less than `r` squared (`r` is the neighbourhood radius, which must be defined earlier in the code) `nb` is included in the result.

Building an Agent-Based Model

Example: Schelling's segregation model

The exhaustive search for neighbour detection can be implemented using python's list comprehension, for example:

```
neighbours = [nb for nb in agents
              if (ag.x - nb.x)**2 + (ag.y - nb.y)**2 < r**2 and nb != ag]
```

Here ag is the focal agent whose neighbours, nb, are searched for.

The if part in the list comprehension measures the distance squared between ag and nb, and if it is less than r squared (r is the neighbourhood radius, which must be defined earlier in the code) nb is included in the result.

Also note that an additional condition nb != ag is given in the if part. This ensures that when nb == ag, i.e. the distance is 0, ag itself is not mistakenly included as a neighbour of ag.

Once we obtain neighbours for ag, we can calculate the fraction of the other agents whose type is the same as that of ag, and if it is less than a given threshold, the position of ag is randomly reset.

segregation.py – part 1

Complete model

```
from pylab import *

n = 1000 # number of agents
r = 0.1  # neighbourhood radius
th = 0.5 # threshold for moving to another neighbourhood

class agent:
    pass

def initialize():
    global agents
    agents = []
    for i in xrange(n):
        ag = agent()
        ag.type = randint(2) # randomly generates either a '0' or a '1'
        ag.x = random() # randomly generates a number between 0.0 and 1.0 for x position
        ag.y = random() # randomly generates a number between 0.0 and 1.0 for y position
        agents.append(ag) # append (i.e. add) the ith agent into the array 'agents'
```

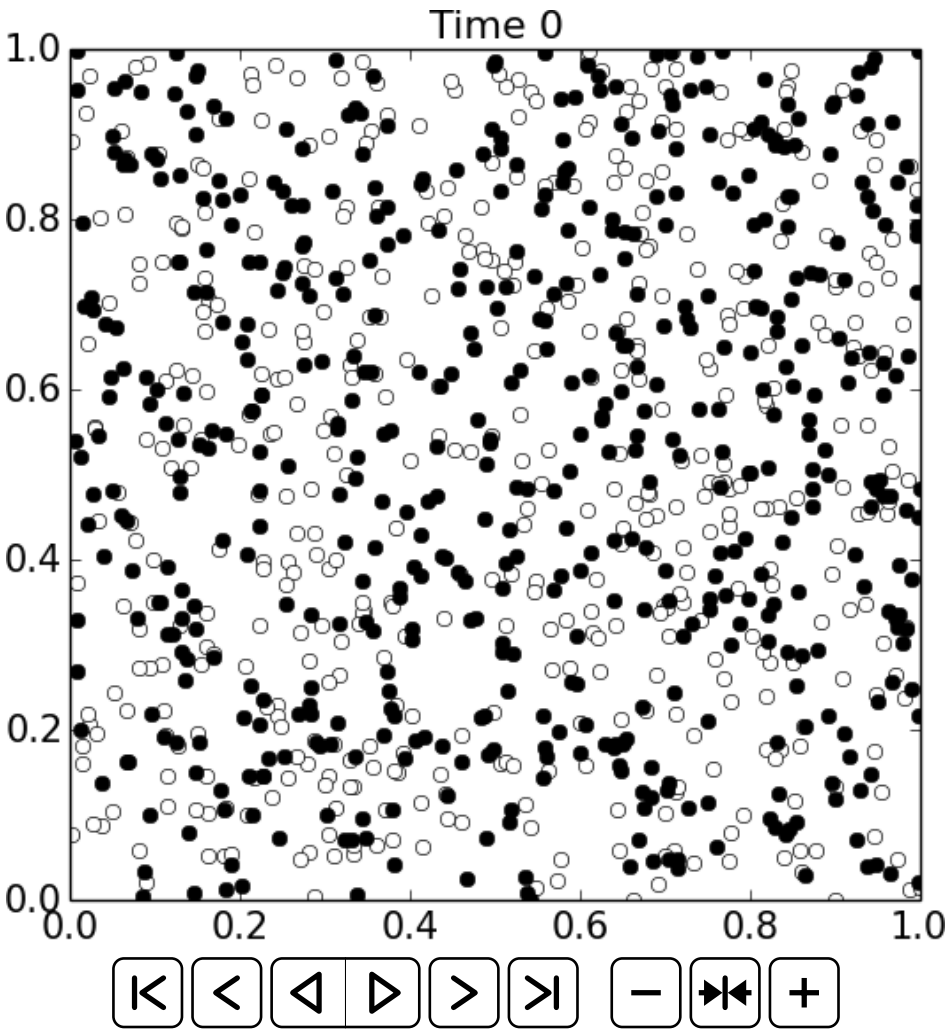
segregation.py – part 2

```
def observe():
    global agents
    cla() # clear axes
    white = [ag for ag in agents if ag.type == 0]
    black = [ag for ag in agents if ag.type == 1]
    plot([ag.x for ag in white], [ag.y for ag in white], 'wo')
    plot([ag.x for ag in black], [ag.y for ag in black], 'ko')
    axis('image')
    axis([0, 1, 0, 1])
    savefig(str(t) + '.png')

def update():
    global agents
    ag = agents[randint(n)]
    neighbours = [nb for nb in agents if (ag.x - nb.x)**2 + (ag.y - nb.y)**2 < r**2 and nb != ag]
    if len(neighbours) > 0:
        q = len([nb for nb in neighbours if nb.type == ag.type]) / float(len(neighbours))
        if q < th:
            ag.x, ag.y = random(), random()

t=0
initialize()
observe()

for t in xrange(1, 100):
    update()
    observe()
```

Results of Shelling's model

This movie shows the result of Shelling's model with a neighbourhood radius of $\mathtt{r} = 0.1$ and a threshold for moving of $\mathtt{th} = 0.5$.

The agents self-organize from an initially random distribution to a patchy pattern where the two types are clearly segregated from each other.



Exercises

Exercise 1 Conduct simulations with the segregation model changing the parameteres `th` (threshold for moving) and `r` (neighbourhood radius). For each value of `r` (0.1, 0.2, 0.3, 0.4, 0.5 0.6, 0.7, and 0.8) determine the threshold at which the model shows segregation and plot these threshold values as a function of `r`.

Exercise 2 Facultative Conduct a sensitivity analysis (see Lecture 8) by developing a metric that characterizes the level of segregation (use for example the index of dissimilarity) once the model has reached a steady-state; then plot how this metric changes as parameter values (e.g. `th`) are varied.

to Further reading

Classes and Objects in Python

Understanding `classes` and `objects`

When you develop an ABM in python, you work constantly with `classes` and `objects`. These two operators are really important.

Understanding `classes` and `objects`

When you develop an ABM in python, you work constantly with `classes` and `objects`. These two operators are really important.

Example:

Think about what it means for something to be a chair.

Understanding `classes` and `objects`

When you develop an ABM in python, you work constantly with `classes` and `objects`. These two operators are really important.

Example:

Think about what it means for something to be a chair.

A chair has a seat, a back, and legs. Each seat has a shape, a color, a degree of softness, and so on. These are the properties of a chair.

Understanding `classes` and `objects`

When you develop an ABM in python, you work constantly with `classes` and `objects`. These two operators are really important.

Example:

Think about what it means for something to be a chair.

A chair has a seat, a back, and legs. Each seat has a shape, a color, a degree of softness, and so on. These are the properties of a chair.

What is described is *chairness* – the notion of something being a chair. In object-oriented terminology, this describes a `class` called chair.

Understanding `classes` and `objects`

Example:

Now take a minute to look around the room.

Understanding `classes` and `objects`

Example:

Now take a minute to look around the room.

Several chairs are in the room, and each chair is an `object`. Each of these objects is an example of a chair class.

Understanding `classes` and `objects`

Example:

Now take a minute to look around the room.

Several chairs are in the room, and each chair is an `object`. Each of these objects is an example of a chair class.

So that's how it works: the class is the description of what a chair should look like, and each individual chair is an object of that class.

Understanding `classes` and `objects`

Definition:

A `class` is simply a *description* of what things should look like, what variables will be grouped together, and what functions can be called to manipulate those variables. Using this description, python can then create many instances (`objects`) of the class, which can then be manipulated independently.

Understanding `classes` and `objects`

Remember:

Understanding `classes` and `objects`

Remember:

A class is the idea behind a certain kind of thing, thus a class is a description of something.

Understanding `classes` and `objects`

Remember:

A class is the idea behind a certain kind of thing, thus a class is a description of something.

When we talk about the class of chairs in a room, we are talking about the fact that each chair has legs, a seat, a color, and so on.

Understanding `classes` and `objects`

Remember:

A class is the idea behind a certain kind of thing, thus a class is a description of something.

When we talk about the class of chairs in a room, we are talking about the fact that each chair has legs, a seat, a color, and so on.

When you talk about a class of things, you are focusing on the type of properties that each of the things possesses not on their actual values.

Understanding `classes` and `objects`

Here is another way to think about a class. Consider the following table displaying your bank accounts.

Understanding `classes` and `objects`

Here is another way to think about a class. Consider the following table displaying your bank accounts.

Table of Accounts

Account Number	Type	Balance
16-132450-00038-7	Checking	147.5
1011 1234 2122 0000	Credit	-470.5
12-143504-00127-5	Savings	257.5

Understanding `classes` and `objects`

Here is another way to think about a class. Consider the following table displaying your bank accounts.

Table of Accounts

Account Number	Type	Balance
16-132450-00038-7	Checking	147.5
1011 1234 2122 0000	Credit	-470.5
12-143504-00127-5	Savings	257.5

Think of the table as a class and think of each row of the table (the single account) as an object.

Understanding `classes` and `objects`

Here is another way to think about a class. Consider the following table displaying your bank accounts.

Table of Accounts

Account Number	Type	Balance
16-132450-00038-7	Checking	147.5
1011 1234 2122 0000	Credit	-470.5
12-143504-00127-5	Savings	257.5

Think of the table as a class and think of each row of the table (the single account) as an object.

Each object in the class `accounts` (that is, each instance of the class `accounts`) has an account number, a type, and a balance (properties of the class).

Understanding `classes` and `objects`

Here is another way to think about a class. Consider the following table displaying your bank accounts.

Table of Accounts

Account Number	Type	Balance
16-132450-00038-7	Checking	147.5
1011 1234 2122 0000	Credit	-470.5
12-143504-00127-5	Savings	257.5

Think of the table as a class and think of each row of the table (the single account) as an object.

Each object in the class `accounts` (that is, each instance of the class `accounts`) has an account number, a type, and a balance (properties of the class).

So, the bottom row of the table is an object with account number 12-143504-00127-5. This same object has type "Savings" and a balance of 257.5.

Understanding `classes` and `objects`

Here is another way to think about a class. Consider the following table displaying your bank accounts.

Table of Accounts

Account Number	Type	Balance
16-132450-00038-7	Checking	147.5
1011 1234 2122 0000	Credit	-470.5
12-143504-00127-5	Savings	257.5

Think of the table as a class and think of each row of the table (the single account) as an object.

Each object in the class `accounts` (that is, each instance of the class `accounts`) has an account number, a type, and a balance (properties of the class).

So, the bottom row of the table is an object with account number 12-143504-00127-5. This same object has type "Savings" and a balance of 257.5.

If you opened a new account, you would have another object, and the table would grow an additional row. The new object would be an instance of the same class account.

Creating a `class` in python

A `class` is a *blueprint*. It is not something in itself, it simply describes how to *make* something. as mentioned such objects are known technically as *instances* of the class.

Creating a `class` in python

A `class` is a *blueprint*. It is not something in itself, it simply describes how to *make* something. as mentioned such objects are known technically as *instances* of the class.

So how do you create these classes ?

Creating a `class` in python

A `class` is a *blueprint*. It is not something in itself, it simply describes how to *make* something. as mentioned such objects are known technically as *instances* of the class.

So how do you create these classes ?

Very easily, with the `class` operator.

Creating a `class` in python

Example: defining a class

We can use a class to create a simple data structure that groups several variables together, e.g.:

Creating a `class` in python

Example: defining a class

We can use a class to create a simple data structure that groups several variables together, e.g.:

```
class Person:
    # An empty call definition
    pass
```

Creating a `class` in python

Example: defining a class

We can use a class to create a simple data structure that groups several variables together, e.g.:

```
class Person:
    # An empty call definition
    pass

>>> individual1 = Person()
```

Creating a `class` in python

Example: defining a class

We can use a class to create a simple data structure that groups several variables together, e.g.:

```
class Person:
    # An empty call definition
    pass

>>> individual1 = Person()
>>> individual1.name = 'Sherlock Holmes'
```

Creating a `class` in python

Example: defining a class

We can use a class to create a simple data structure that groups several variables together, e.g.:

```
class Person:
    # An empty call definition
    pass

>>> individual1 = Person()
>>> individual1.name = 'Sherlock Holmes'
>>> individual1.address = '221B Baker Street'
```

Creating a `class` in python

Example: defining a class

We can use a class to create a simple data structure that groups several variables together, e.g.:

```
class Person:
    # An empty call definition
    pass

>>> individual1 = Person()
>>> individual1.name = 'Sherlock Holmes'
>>> individual1.address = '221B Baker Street'
>>> individual1.phone = '+40 20 7452 339'
```

Creating a `class` in python

Example: defining a class

We can use a class to create a simple data structure that groups several variables together, e.g.:

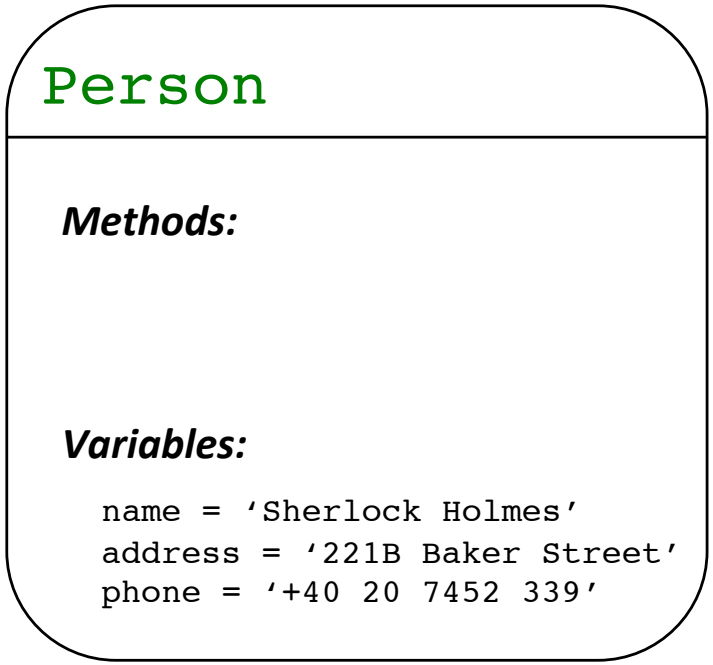
```
class Person:
    # An empty call definition
    pass

>>> individual1 = Person()
>>> individual1.name = 'Sherlock Holmes'
>>> individual1.address = '221B Baker Street'
>>> individual1.phone = '+40 20 7452 339'
```

So, we created an empty class definition, then took advantage of the fact that python will automatically create member variables when they are first assigned.

Creating a `class` in python

Example: defining a class

The object `individual1` can be represented with the following diagram:



back to agents

List Comprehension in Python

List comprehension in python

Python supports a concept called "list comprehension". It can be used to construct lists in a very natural and easy way, like a mathematician is used to do.

The following are common ways to describe lists (or sets, or tuples, or arrays) in mathematics:

$$\mathtt{S} = \{\mathtt{x}^2 \ \forall \ \mathtt{x} \ \in \ \{0, \ldots, 9\}\}$$
$$\mathtt{V} = \{1, 2, 3, 4, 8, \ldots, 2^{12}\}$$
$$\mathtt{M} = \{\mathtt{x} \ \forall \ \mathtt{x} \ \in \ \mathtt{S} \ \mathtt{and} \ \mathtt{x} \ \mathtt{even}\}$$

List comprehension in python

Translated in "normal" language, this means:

Consider a list S with x^2 for all x in $\{0, \ldots, 9\}$

Consider a list V with 1, 2, 4, 8, . . ., 2^{12}

Consider a list M with x for all even x in S

This is how you do the above in python:

```
>>> S = [x**2  for x in range(10)]
>>> V = [2**i  for i in range(13)]
>>> M = [x  for x in S  if x%2 == 0]
>>>
>>> print S; print V; print M
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096]
[0, 4, 16, 36, 64]
```

List comprehension in python

The list comprehension starts with `[` and ends with `]`, to help you remember that the result is going to be a list.

The basic syntax is:

```
result = [ transform      iteration          filter ]
```

For example:

```
result = [   x**2     for x in range(5)     if x%2 == 0   ]
```

Which is equivalent to:

```
result = []
for x in range(5):
     if x%2 == 0:
          result.append(x**2)
```

back to model

Further reading


Hiroki Sayama 2015
Introduction to the Modeling and Analysis of Complex Systems
Open SUNY Textbooks


Volker Grimm & Steven F. Railsback 2005
Individual-based Modeling and Ecology
Princeton University Press


Robert Axelrod 1997
The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration
Princeton University Press


A. N. Macal & M. J. North
Tutorial on agent-based modelling and simulation
Journal of Simulation, 4:151–162, 2010


E. Bonabeau
Agent-based modeling: Methods and techniques for simulating human systems
PNAS, 99(3):7280–7287, 2012


T. C. Schelling
Dynamic models of segregation
Journal of Mathematical Sociology, 1(2):143–186, 1971