

What Is Programming?

Programming is the process of writing the code of computer programs. A program is just a sequence of instructions that a computer is able to read and execute, to make something happen. Like opening a window on a desktop.

In order for a computer to read and execute code, it ultimately has to be given a string of binary code, 1's and 0's. At the machine level, the code looks like this: 001010010110100110101001011010. Typically, however, people write code that looks something like this:

```
>print "hello" [a command in the Python programming language]
```

How do you get from such language and symbols that we understand to symbols and numbers the computer chip can process? Through what is called a compiler, which breaks down the code we write into a more machine-like language called assembly language. This in turn is translated by a piece of software called an assembler into machine language. Thus the steps are:

write code > compiler translates code > outputs assembly language > assembler translates code > outputs machine language (0's and 1's).

Except for the first step, all of those steps happen under the hood and we don't need to be concerned with them here.

Different programming languages have different syntaxes, like the difference between English and Latin. For example, in English

```
dog loves lion  
lion loves dog
```

are two utterly different sentences because word order signals whether a noun is a subject or object of a sentence. In Latin,

```
canis amat leonem  
leonem amat canis
```

are the *same* sentence, both equivalent to 'dog loves lion' because Latin is largely agnostic about word order.

To get the second sentence, lion loves dog, in Latin, you'd write either:

```
canem amat leo  
or  
leo amat canem
```

Overview of Programming and Max/MSP/Jitter

To pair them another way:
canis amat leonem
canem amat leo

In programming, order will matter both within individual statements and within groups of statements. Thus, while

```
>chmod 777 index.html
```

is a meaningful, executable piece of code in the Unix programming language

```
>777 chmod index.html  
>index.html 777 chmod  
>chmod index.html 777
```

are not and will result in error messages.

What is Max?

Max is a graphical programming environment originally developed by Miller Puckette, a professor at UCSD. Work on it began in 1986 and it became a commercial product released by Cycling '74 in 1991. It was designed to work as a controller and programmer for MIDI devices. (MIDI: musical instrument digital interface: a standard protocol for communication between electronic musical instruments and computers.) Max is based on the C programming language, but this invisible behind the GUI. From release 4.5, Max also supports Java and Javascript.

Since Max is visual programming language, you build your code not by writing strings of text that looks more or less like gibberish, but by connecting together icons. Your code may still break because the logic of the connections is flawed, but at least you won't have to hunt through hundreds of lines of codes to see where you forgot to insert a semi-colon.

Max has a library of about 200 objects (functions). It has since been extended by two further libraries, MSP and Jitter.

MSP is a library of about 200 extra Max objects you can use for audio signal processing, including building your own software versions of sound synthesizers, music samplers, and effects processors. We won't be doing a great deal with MSP in this class.

Jitter is a different library of about 140 Max objects oriented mostly towards working with video, still images, and 3D images. Jitter is useful to for realtime video processing, as well as audio/visual interaction.

Working in Max

Programs that you write in Max run in real-time; in effect, they are compiled on the fly (more about this below). Anyone can run a Max program using a free downloadable runtime program from Cycling '74. However, writing in Max requires buying the Max/MSP/Jitter software from Cycling '74, and even the student versions are not cheap.

One-month trial versions of MMJ are free and can be downloaded from Cycling '74. I suggest that if you want to get a trial version so that you can work at home and not just in the lab, you wait to start your trial subscription until towards the end of the quarter, so that you can work on your final project. There are a number of different versions of Max/MSP/Jitter available; be sure to get the right one for your OS. Also, since Max/MSP and Jitter are separately downloaded, and have separate version-numbering systems, make sure you download mutually compatible versions.

There are a pair of main manuals for each program. The tutorial manual includes detailed descriptions of various functions, with examples. Tutorial-related patch files can be found in a separate folder, allowing you to interact directly with the examples in each chapter of the tutorial. The reference manual lists all the major objects of either Max or Jitter with critical information about kinds of input and output, and some examples. It is a good source to consult when an object isn't working as you expected, or when you want to learn more about an object you've never used before.

Manual conventions: objects are in **bold text**, messages are in plaintext.

Max Syntax

In Max, the syntax consists not of primarily nouns and verbs (as in English) but of **objects** and the **messages** that pass between objects by way of **patch cords**.

Let's return to

```
>print "hello" [Python]
```

What does this look like in Max?



The two boxes are different types of objects, as can be told from their different borders (there are other types as well that we will learn). The lower box is an **object** box; in this case a **print** object, which prints to the Max window whatever message it receives by way of the patch cord coming in to the top edge (the **inlet**). The print object is what in

Overview of Programming and Max/MSP/Jitter

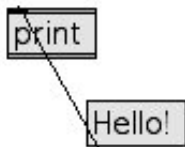
other programming languages is called a function. A function is like a mini-program that is built in to the programming language (library) so that you don't have to write it from scratch yourself. The upper box is a **message** object, which passes whatever is typed inside it to other objects via a patch cord coming out of its lower edge (the **outlet**).

Inlets and outlets are shown as blackened bars on the rim of the box. Inlets are always on top, and outlets are always on the bottom.

What happens is this: clicking the message box sends the message shown to the print box, which prints it to screen in the Max window.



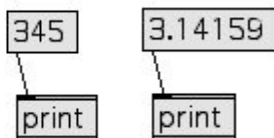
Syntax: if the order of the boxes were reversed, with the print box above the message box, you'd find that you could not connect the two boxes in the same pattern as above—with a cord from the lower edge of the top box to the upper edge of the lower box. Messages flow from message objects to print objects and not vice versa. The program won't even let you make this mistake because the print box has no outlet (by default it prints to the Max window—in effect, the outlet is there but invisible to us and not manipulable by us).



You can, however, move the boxes around on screen and as long as they are connected *the exact same way*, the program will work. When there is only a single patch cord, placement on screen is a matter of preference.

In Max, pretty much everything is stored in one box or another. You'll find that different boxes have different numbers of inlets and outlets, where information flows in and out.

Also, the message box can store different kinds of information. In the examples above, we're using it to store text. It can also store numbers.



In the example at left, the box is storing an integer, called `int` in Max. At right, it's storing a fraction or floating-point number, called a `float` in Max. Other kinds of boxes exist that

can only store ints or floats, and other kinds of messages can be stored and passed, including lists.

When you connect Max objects with patch cords, you are connecting functions written in C that already exist "under the hood" and have been precompiled. In essence, you are creating the links that stitch together a bunch of miniature code modules.

Interface Basics

--> NOW LAUNCH MAX<--

When you launch Max, two crucial windows appear: the edit window and the Max window. The edit window, of course, is where you write your programs; and the Max window serves as a test window, showing output from functions whose output is textual or numerical. Error messages also show up in the Max window. There are other windows, like the Inspector, that we will learn about as we go along.

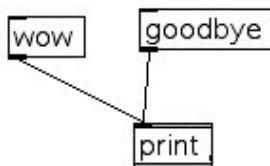
1. recreate the "print hello" program from scratch. run it.

When you start to type in the word "print" a list comes up showing Max's built-in functions; you can scroll down the list to choose the function rather than type it in.

Note that even if you create the program exactly right, it won't run in the edit window. In order to run the program, you must switch from edit to run mode by clicking on the small oval button in the upper right corner of your edit window. Note that the appearance of the window changes when you do this; for example, the button menu disappears. Now you can run the program. The oval button is a toggle that allows you to switch between edit and run at any time.

2. make the program print GOODBYE by editing the message box.

3. make the program print SOMETHING ELSE by creating and linking a new message box without disconnecting the first.



You've just learned that an object's inlet box can have multiple messages connected to it.

4. create comment boxes for the program.

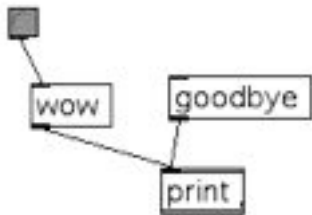
Note that comment boxes cannot be attached by patch cords to anything because they are nonfunctional. They are labels. They can be resized by dragging on the lower right corner.

5. save the program with a one-word name to make a patch

Using the Bang and Metronome

The bang is a special type of message, that when activated, essentially says, "do whatever you are supposed to do" or "go". Think of it as a trigger.

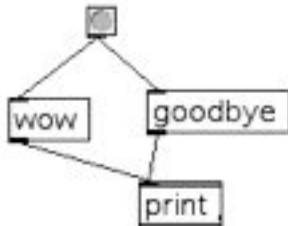
1. add a bang box and connect it to the left inlet of one of the message boxes (note that it does not affect the second message box in any way).



The bang message just tells the message box to do what it normally does, which is send its message to the print object.

Whenever you are uncertain about how an object functions in Max, you can option-click on it to have a help window pop up. Further information can be found in the Max/MSP/Jitter Reference Manuals.

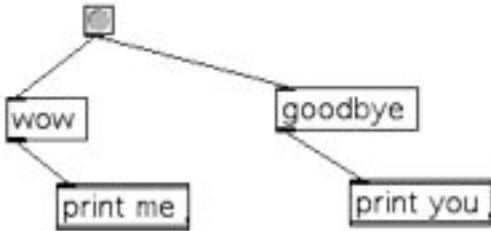
2. connect the bang box to the second message box.



Note that the message is printed **from right to left**. This is always the case in Max.

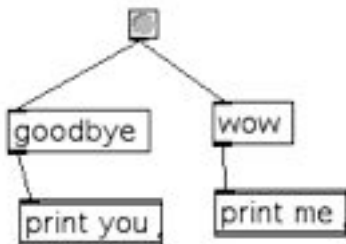
Note also that both messages appear undifferentiated in Max window-- you can't tell them apart. By giving the print object what is called an **argument**, they can be distinguished from each. An argument is additional information that modifies the object in some way or specifies certain optional characteristics. For the print object, the only possible argument is a name.

3. disconnect the righthand message box from the print object and connect it to its own print object. keep it connected to the bang box. give the two print objects their own names, which should be a single word.



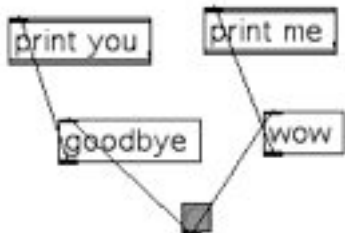
You can speed up this process by selecting, copying, and pasting the print box (rather than creating it from scratch from the menus). Note that the names of the print objects (here, "you" and "me") now appear in the Max window.

4. now reverse the message/print object pairs on screen



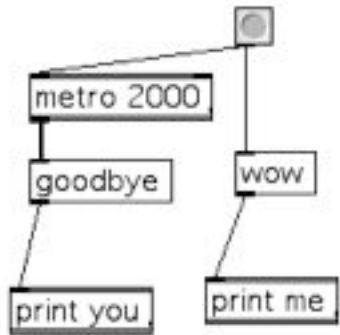
Note that the objects are now printing *in reverse order*. This is because, unlike our earlier example, where there are multiple objects attached to the same message, they will always be dealt with from right to left. In this case, horizontal placement does matter.

Vertical placement doesn't strictly matter, but recall that inlets have been placed at the tops of objects and outlets along the bottom. For this reason, it is always easier to understand your code both visually and structurally if you construct it from the top of the window going down.



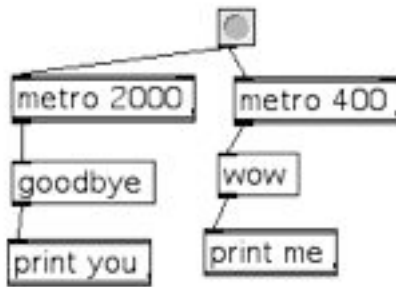
The **metronome** object is like a bang button, but where the bang sends its "go" once and stops, the metronome keeps sending "go" at regular intervals.

1. create a metro object. set its argument to 2000 (milliseconds). connect it between the bang and one of the message boxes.



With the metronome object, the argument is a setting in milliseconds of the interval between bangs. It defaults to 1000 if you don't set anything. Note that one message is only printed a single time, while the other prints repeatedly-- unless you hit the bang again.

2. add a second metro object with a different argument between the bang and the other message box.



Note the pattern that appears in the Max window.

The metronome can take a number of different kinds of inputs in its left input box.

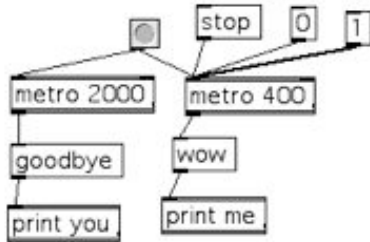
3. create a message box containing the word 'stop' and add it to the input of one of the metro objects.

Note first what happens when you click on the stop message box: the word "stop" does not appear in the Max window even though there is a print object further down the chain. However is this message box different from the earlier "goodbye" message box we made? The stop message operates differently when it is the input to a metro object; in this case it tells the metro object to stop passing its own messages further down the chain. Max calls this kind of message a **symbol**. You can test this by adding a second stop box to the input of either print box; you will see it functions as expected in this case.

Note also what happens when you switch between the stop and the bang.

4. add two more message boxes to the input of one of the metro objects. these boxes should contain a 0 and a 1.

The 0 and the 1 work like stop and bang. Shortly, we will learn about a toggle button that replaces these boxes.

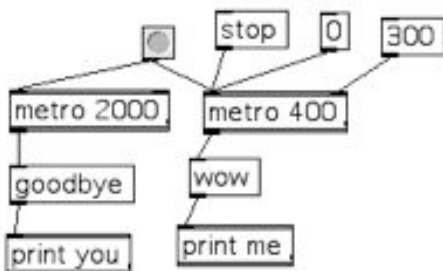


5. change the 1 box to contain the number 300.

The metronome will start if it receives any nonzero number and stop if it receives a 0.

The metro object has 2 inlets. Messages into the **left** inlet can only stop or start the metronome in various ways. Messages sent into the **right** inlet will change the argument of the metronome-- that is, change the metronome interval.

6. disconnect the 300 box from the left inlet and attach it to the right inlet.

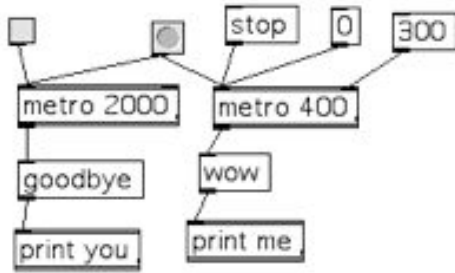


Note that the pace of that message changes even though the visible argument number doesn't change. The metro box will always show the original argument setting even when the argument is changed through the right inlet. Later we will learn ways to show the current state of a given object.

The Toggle Button

1. attach a toggle button to the left inlet of the metro object.

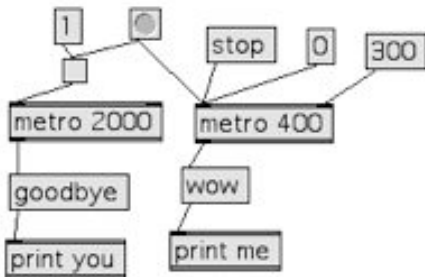
The toggle alternately sends out a 0 or a 1 when clicked. Initially it is set up to send out a 1 (start something running). Once it sends out the 1, it shows an X in the box, signalling that the next time it is used it will send out a 0 (stop). When it sends out a 0, it reverts to its initial blank stage, signalling that the next time it is clicked it will send out a 1 (go). In effect, it is an on-off switch.



2. now insert the toggle between the bang and the metro object.

What is the difference?

3. now create a new message box with the number 1 inside and attach it to the inlet of the toggle box.



What is the difference?

CLASS EXERCISE

Create a group of 5-10 message boxes each containing a segment of text—one word or a few words. Connect these message boxes to some print boxes (with no arguments). Experiment with different combinations of metro objects (and metro argument change boxes), bangs, toggles, stop boxes, and 0/1 boxes connected to the message boxes. Your goal is to create a poetry jam that is partly automated (through the metro object) and partly user controlled (through 'playing' the control buttons). Try it through the Mac's text-to-speech processor.

Tips:

- some print boxes can have multiple message boxes attached to them.
- a combination of nouns, verbs, adjectives will create the most interesting results
- think about what kinds of words or phrases you want to repeat often, and what kinds you want to appear infrequently.