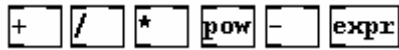


# Pure Data

---

**MATH OBJECTS** are for calculating mathematic equations in Pd. Here are a few of the most common ones needed to start programming in Pd.



---

## **ADDITION**

**Left Inlet:** The number which is sent to the left inlet is the number to be added. Sending a number to the left inlet results in a calculation. Sending a **BANG** to the left inlet also results in an action taking place.

**Right inlet:** The number sent to the right inlet is stored for addition to a number received in the left inlet.

**Arguments:** Arguments can be floats or integers. The argument is added to the number which is sent to the left inlet.

**Output:** A number will be outputted anytime the left inlet is activated. The output can be either a float or an integer value depending on the calculation that took place.

## **SUBTRACTION**

**Left input:** The number which is sent to the left inlet is the number from which a subtraction will take place. Sending a number to the left inlet results in a calculation. Sending a **BANG** to the left inlet also results in an action taking place.

**Right inlet:** The number sent to the right inlet is stored for subtraction from a number received in the left inlet.

**Arguments:** Arguments can be floats or integers. The argument is the number from which a subtraction will take place.

**Output:** A number will be outputted anytime the left inlet is activated. The output can be either a float or an integer value depending on the calculation that took place.



## MULTIPLY

**Left input:** The number which is sent to the left inlet is the number which to be multiplied. Sending a number to the left inlet results in a calculation. Sending a **BANG** to the left inlet also results in an action taking place.

**Right inlet:** The number sent to the right inlet is stored for multiplication to a number received in the left inlet.

**Arguments:** Arguments can be floats or integers. The argument is the number which will be multiplied by the number which is received through the left inlet.

**Output:** A number will be outputted anytime the left inlet is activated. The output can be either a float or an integer value depending on the calculation that took place.



## DIVISION

**Left input:** The number which is sent to the left inlet is the number to be divided. Sending a number to the left inlet results in a calculation. Sending a **BANG** to the left inlet also results in an action taking place.

**Right inlet:** The number sent to the right inlet is stored and used as a divider to a number received in the left inlet.

**Arguments:** Arguments can be floats or integers. The argument is the number which will divide the number which is received through the left inlet.

**Output:** A number will be outputted anytime the left inlet is activated. The output can be either a float or an integer value depending on the calculation that took place.



## POWER OF

**Left input:** The number which is sent to the left inlet is the number to be raised by a particular power. Sending a number to the left inlet results in a calculation. Sending a **BANG** to the left inlet also results in an action taking place.

**Right inlet:** The number sent to the right inlet is stored and used as a power coefficient to the number received in the left inlet.

**Arguments:** Arguments can be floats or integers. The argument is the the power coefficient

**Output:** A number will be outputted anytime the left inlet is activated. The output can be either a float or an integer value depending on the calculation that took place.



**EXPRESSION** –evaluates a mathematical expression

**Inlets:** The number received in each inlet will be stored in place of the \$f argument associated with it. (Example: The number in the second inlet from the left will be stored in place of the \$f2 argument, wherever they appear.)

**Arguments:** Arguments are Obligatory. The argument is a mathematical expression, in a format resembling the C programming language. The expression is made up of numbers, arithmetic operators such as + or \*, comparisons such as < or >, C functions such as min() or pow(), names of table objects, and changeable arguments (\$i, \$f, and \$s) for ints, floats, and symbols received in the inlets.

**Example:**

```
expr 2+$f1;
```

This will output 2+ whatever number is inputted to the expr object.

**Output:** The output is the result of the evaluated expression.

**GLUE OBJECTS** are used time and time again in Pd to do a variety of task including comparisons, tests, sorting, etc. These are a few of the most common ones to get you started.

`float` `int` `select` `print`

---

`float` **FLOAT** -stores a float value

`int` **INT** -stores an integer value

**Left input:** The number which is sent to the left inlet replaced the previously stored number and outputs it. Sending a number to the left inlet results in an action taking place. Sending a **BANG** to the left inlet also results in an action taking place.

**Right inlet:** The number sent to the right inlet is stored and can be recalled by activating the left inlet.

**Arguments:** Arguments can be floats or integers depending on the object. The argument is the initial value to be stored. If there are no creation arguments, the initial value will be 0.0 or 0 according the object.

**Output:** A number will be outputted anytime the left inlet is activated. The output can be either a float or an integer value depending on the object.

`select` **SELECT** -used to test incoming data and output a result accordingly

**Left inlet:** The left inlet accepts any message. If the input matches one of the arguments, a bang is sent out the outlet that corresponds to that argument. Otherwise, the input is passed out the rightmost outlet.

**Right inlet:** The right inlet replaces the value of the creation arguments. The right inlet exists only if there is a single '**int**' argument.

\*A bang in the left inlet will converted to a symbol bang and treated as any other symbol.

**Arguments:** The arguments can be a mix of ints, floats, or symbols. The number of arguments determines the number of outlets in addition to the rightmost outlet. If there is no argument, there is only one other outlet, which is assigned the integer number 0.

**Output:** If the number or symbol received in the left inlet is the same as one of the

arguments, a **BANG** is sent out the outlet that corresponds to that argument. If the number or symbol received in the left inlet does not match any of the arguments, it, the number or symbol is passed out the rightmost outlet.

**print** **PRINT** –prints a message to the command prompt(in the latest version of Pd the prompt is integrated to the main pd window)

**Left input:** Any message can be inputted to the left inlet.

**Arguments:** The optional argument is an identifier for the print object. Each message printed will be preceded by the name of the print object, and a colon (:). The name must not contain spaces or special characters, but can be either a number or a word. If there is no argument, the name of the print object is print. Using an argument to print can help distinguish the output of two or more print objects and can be very useful when debugging complex patches.

**Output:** There are no outlets. The message received in the inlet is printed directly to the command prompt.

**MIDI OBJECTS** are used to interact with Midi interfaces; here are a few to get you started.

`notein` `makenote` `noteout` `pgmout`

---

`notein` **NOTEIN** –receives note information from a MIDI device

**Arguments:** NOTEIN can take an optional argument that specifies a particular channel number. With this set NOTEIN will only accept information from that channel.

**Output:** The first outlet outputs a integer pitch anytime a note is activated on the MIDI device. The second outlet output the velocity value of the note and the third output outputs the channel that the NOTEIN object is receiving the information from.

\*If no channel number is set the NOTEIN object defaults to OMNI, which means it will receive information from all MIDI channels.

`makenote` **MAKENOTE** –constructs a midi note message from integer data

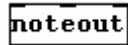
**Left inlet:** The number sent to the left inlet is treated as a pitch value for a MIDI note-on message. It is then paired with a velocity value (middle inlet) and the numbers are sent out the outlets. After a certain time (determined by the right inlet), a note-off message (a note-on with a velocity of 0) is sent out for that pitch.

**Middle inlet:** The middle inlet receives a velocity mapping for the note received through the first inlet.

**Right inlet:** This inlet receives an integer duration in milliseconds for which the note will sound.

**Arguments:** Optional arguments can be used to specify initial velocity and duration

**Output:** The number received in the left inlet is sent out immediately, paired with a velocity value which is sent out the right outlet. After a certain duration, the same number is sent out paired with a velocity of 0.



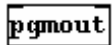
**NOTEOUT** –outputs MIDI note information, most often used with a [MAKENOTE] object.

**Left inlet:** The number received through the left inlet is the pitch value of a MIDI note message transmitted on the specified channel and port. Numbers are limited between 0 and 127.

**Middle inlet:** The number received through the middle inlet is stored as the velocity of a note message, to be used with pitch values received in the left inlet. Numbers are limited between 0 and 127. 0 is considered a note-off message, 1-127 are note-on messages.

**Right inlet:** The number received through the right inlet is stored as the channel number on which to transmit the note-on messages.

**Arguments:** Optional arguments can be used to specify the MIDI channel you wish to use.

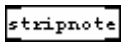


**PGMOUT** –outputs a program change to a MIDI device

**Left inlet:** The number sent to the left inlet is transmitted as a program change value on the specified channel and port. Numbers are limited between 1 and 128, and are sent out as program changes 0 to 127. Therefore if you send a 1 to the [PGMOUT] object it will send a program change value of 0

**Right inlet:** The number sent to the right inlet is stored as the channel number on which to transmit the program change messages.

**Arguments:** An optional argument can be used to set the channel number on which to transmit the program change.



**STRIPNOTE** –strips note-off messages from a MIDI message

**Left inlet:** A MIDI pitch is sent to the left inlet where only the note-on message is allowed through..


**Right inlet:** A MIDI velocity is sent to the right inlet. The number sent to the right inlet is stored as the channel number on which to transmit the program change messages.

**Outputs:** The MIDI note number is outputted its left outlet and a MIDI velocity is outputted its right outlet.

**TIME OBJECTS** allow users to control all aspects of timing in a patch; here are a few of the most basic ones.

---

 **METRO** –metronome object

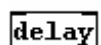
**Left inlet:** The left inlet receives a number or bang. Any number other than 0 starts metro. A 0 in the left inlet will stop the metro.

**Right inlet:** This inlet receives the time interval duration in milliseconds. For example if the number 1000 is sent to this inlet the METRO will bang every second.

**Arguments:** An optional argument can be used to set the initial value for the time interval.

**Output:** A bang is sent immediately when the METRO starts and subsequently every ‘n’ milliseconds after the initial bang.

\*If no time interval is passed to the metro object it will be set to 5 milliseconds by default.

 **DELAY** –delays a bang by a given duration before passing it on

**Left inlet:** The left inlet receives a bang which will then be delayed by a given duration. The left inlet may also take a STOP message which will prevent the delay from carrying out its task.

**Right inlet:** This inlet receives the time interval duration in milliseconds. For example if the number 1000 is sent to this inlet all bangs which are received in the left inlet are delayed by 1 second before being outputted..

**Arguments:** An optional argument can be used to set the initial value for the time interval.

**Output:** A bang is sent after the given delay time.





**TIMER** –calculates elapsed times between particular events

**Left inlet:** The left inlet receives a bang which will start the timer

**Right inlet:** When the TIMER receives a bang to its right inlet it will stop the timer and output the result.

**Arguments:** None

**Output:** The total amount of time which elapsed between bangs to the left and right inlet is outputted in milliseconds.

\*TIMER is one of the few exceptions where the right inlet is *hot* and the left inlet is *cold*

**MISCELLANIOUS OBJECTS** fall under no particular category. They include objects that allow easy interaction with users, objects that allows uses to open and save files, object that allow interaction with serial devices and much more. For now I only want to show you these few simple MISC objects which allow you to grab whatever's being typed on the keyboard.

**key** **keyup** **keyname**

---

**key** **KEY** –receives information from keyboard

**Output:** An output is sent each time a key is pressed on the computer keyboard. (Holding the key down does not produce repeated output.) The output sent is the ASCII value of the typed key.

**keyup** **KEYUP** –receives information from keyboard

**Output:** An output is sent each time a key is **de**-pressed on the computer keyboard. The output sent is the ASCII value of the typed key.

**keyname** **KEYNAME** -receives information from keyboard

**Output:** An output is sent each time a key is pressed on the computer keyboard. A '1' is sent through the left outlet when the key is pressed and is followed by a 0 whenever the key is de-pressed. The key character name is sent out the other outlet.

**SUBPATCH OBJECTS** are objects that allow the user to organise the different sections of their patch into sub-patch, or hidden windows. When used correctly these objects help to make patches user friendly and intuitive for the user.

**pd** **inlet** **outlet** **table**

---

**pd** **PD** –creates a Pd subpatch

**Input:** Any type of input is possible. The numbers of inputs will be determined by the number of inlet objects (*see below*) contained in the patch.

**Outputs:** Any type of output is possible. The numbers of outputs will be determined by the number of outlet objects (*see below*) contained in the patch.

**Arguments:** An optional name can be given to the subpatch. This name will appear when the user enters the subpatch from the main patch. Naming subpatch is highly encouraged as it gives a clearer structure to the overall layout of the patch.

**ABSTRATIONS** –*abstractions are similar to subpatches. They can be evoked by typing the name of a patch previously saved into an object box. If the path to the patch is the same as the current working directory nothing needs to be done to get it to run, if the patch is saved in a different directory the ‘path’ must be specified to Pd on startup. Changing the abstraction means that all subsequent instances of it will also be changed*

**GRAPH-ON-PARENT** –*graph on parent allow you to control your subpatch GUI controls(sliders, number boxes, radio-check boxes....etc) from the parent patch. By right clicking on the canvas in a subpatch or abstraction you can click on the graph-on-parent checkbox. Now when you close the abstraction/subpatch you can control those GUI elements from the parent patch.*

**inlet** **INLET** –receives input from parent patch

Each inlet object in a patcher will show up as an inlet at the top of the Pd object in the parent patch. Messages sent into such an inlet will be received by the inlet object in the subpatch.

**outlet** **OUTLET** –sends output from subpatch to parent patch

Each outlet object in a patcher will show up as an outlet at the bottom of the Pd object in the parent patch.

**table** **TABLE** –builds a subpatch with a graphical array inside.

**Arguments:** Optional arguments can be used to set the ‘name’ and ‘size’ of the array

You can also send messages to the array by name:

```
;tablename read table.txt [ ;tablename write /tmp/table.txt [
```

*where ‘tablename’ is the name of the table*

---

## MORE TIME OBJECTS

**pipe** **line** **realtime**

---

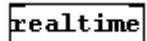
**pipe** **PIPE** –delays a number of list

**Left inlet:** Any type of input is possible. The inputted numbers or list is delayed a certain amount of time before being sent out the output.

**Right inlet:** Accepts a delay time in milliseconds.

**Output:** The messages or numbers that were sent to the left inlet are outputted in the order they were sent.

**Arguments:** An optional name can be given to specify the initial delay time in milliseconds


 **REALTIME** –measures elapsed time from the Operating System

**Left inlet:** The left inlet receives a bang which will start the clock

**Right inlet:** When ‘Realtime’ receives a bang to its right inlet it will stop the clock and output the result.

**Output:** The total amount of time which elapsed between bangs to the left and right inlet is outputted in milliseconds.

\*This object is another example where the right inlet is *hot* and the left inlet is *cold*

 **LINE** –outputs numbers in a ramp from one number to another


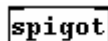
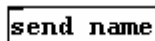
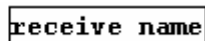
**Left inlet:** The left inlet receives a message with the ‘*initial value*(optional)’, *target*, and *time* in milliseconds.

**Arguments:** The first argument specifies the initial value for line. The second argument specifies the initial ‘grain’, i.e. the time interval between values.


**Output:** The total amount of time which elapsed between bangs to the left and right inlet is outputted in milliseconds.

---

## MORE GLUE OBJECTS

---

 **MOSES** –parts a stream of numbers

**Left inlet:** Any type of input is possible. The inputted numbers to the left inlet are sent out the left or right outlet depending on the range of the numbers.

**Right inlet:** The number sent to the right inlet specifies the maximum value that can be sent out the left outlet.

**Left outlet:** Any number that is below the number specified in the right inlet is passed out the left outlet

**Right outlet:** Any number that is greater than the number specified in the right inlet is sent to this outlet.

**Arguments:** An optional argument can be used to specifies the number t which the parting will take place.

**spigot** **SPIGOT** –acts as a gate for incoming messages

**Left inlet:** The left inlet can receive any type of input which will be passed directly to the output depending on whether the gate is open or closed

**Right inlet:** The right inlet will open the gate when it receives a & and close the gate when it receives a 0

**Output:** The output will depend on whether or not the gate is open, if so the output will match the input, if not, there will be no output.

**send name** **SEND** –sends messages without patch chords

**Inlet:** The inlet receives the message that you want to send

**Arguments:** Send only takes one argument and that is the send objects name (identifier)

*\*SEND must always be used with a RECEIVE object*

**receive name** **RECEIVE** –receives messages without patch chords

**Output:** The output outputs whatever messages that are being sent with the ‘send’ object.

**Arguments:** Receive only takes one argument and that is the objects name (identifier)

*\*RECEIVE must be used with a SEND object*



-normalize  
-rate <sample rate>

## MORE GLUE OBJECTS

**route** **pack** **unpack** **trigger**

---

**route** **ROUTE** –routes messages according to the first element

**Inlet:** Any type of input is possible. The first element of the inputted message acts as an identifier and if the first element of the inputted message matches any of the objects creation arguments it will be passed to the output.

**Outlets:** The number of outlets is determined by the number of argument passed at creation time.

**Arguments:** Arguments determine which outlet the incoming data is passed. If a match is found, the rest of the message appears on the corresponding outlet. If no match, the message is repeated to the last "rejection" outlet. The number of outlets is the number of arguments plus one. The last outlet is the 'rejection' outlet.

**pack** **PACK** –takes a number of atoms and pack them into a single message

**Inlets:** Any type of input is possible. Each inlet is packaged together with the other incoming data into a single message. A 'bang' can be sent to the left most inlet in order to pack the elements together without changing the value of the left element in the new message.

**Outlets:** A single message is outputted.

**Arguments:** Arguments determine how many elements will be packed together. The type of argument determines what type of input is expected through the respective inlets.

**unpack** **UNPACK** –splits a message up into its different

**Inlet:** Message consisting of different elements to be split up.

**Outlets:** Each outlet will output the corresponding inputted message element.



**Arguments:** Arguments determine how many elements will be split into different messages. The type of argument determines what type of input is expected through the respective inlets.

**trigger** **TRIGGER** –sequences messages in a right to left order/can also be used to convert elements.

**Inlet:** Accepts any types of input and outputs them in reverse order.

**Outlets:** Inputted elements in right to left order.

**Arguments:** Arguments determines what type of message will be outputted.

*Trigger can be abbreviated to 't'*

---

## MORE MISCELLANIOUS OBJECTS

**qlist** **textfile**

---

**qlist** **QLIST** -a text based sequencer

**Inlet:** Qlist accepts a series of messages which instruct qlist as to which action to take. Here is the list of message that it can accept.

**'read'** – reads a file

**'bang'** –outputs the contents of the textfile sequentially

**'rewind'** –rewinds the textfile to the beginning again

**'tempo'** –sets relative tempo, therefore sending a 2 will double the speed of playback.

**'next'** –sending a next message will iterate through the contents of each line.

**'add'** –adds some text to the qlist

**'add2'** –adds some text but does not termite, i.e., it does not add a carriage return to the string

**'write'** –writes the list to a textfile

**'clear'** –clears the list

**'print'** –prints the entire contents of the qlist file

**Right outlet:** The right outlet will output a bang when the list has been read.

**Left outlet:** ?

**textfile** **Textfile** –reads and writes to a text file

*This object works in the same way as the previous object only you cannot output data sequentially by specifying a time interval.*

---

## Introduction to TILDE OBJECTS

**dac~** **adc~** **osc~** **phasor~** Tilde objects are used exclusively for dealing with audio signals. They cannot be connected to non-tilde objects. Some of the glue objects that we have seen have a ‘tilde’ equivalent. These objects behave in the same way as their non-tilde counterparts and will not be discussed here.

---

**adc~** **ADC~** –reads audio stream from the soundcard

**Outlets:** By default this object outputs stereo signals

**Arguments:** You can pass an argument to specify the actual audio port that you want to access. For example if you pass one it will open the first audio port available.

**dac~** **DAC~** -outputs audio stream to soundcard

**Inlets:** By default this object receives a left and right channel from a stereo signal

**Arguments:** Again as in the case of **adc~** you can pass an argument to specify the actual audio port that you want to output to.

**osc~** **OSC~** -this is a simple cosine tone generator

**Left Inlet:** Receives a number which determines the frequency to be outputted.

**Right Inlet:** Sets the phase of the cosine

**Arguments:** An initial argument can be used to set the initial frequency

**phasor~** **PHASOR~** -outputs a sawtooth waveform

**Left Inlet:** Receives a number which determines the frequency to be outputted.

**Right Inlet:** Sets the phase of the phasor

**Arguments:** An initial argument can be used to set the initial frequency