

# Asymptote Reference Card

## Program structure/functions

<code>import "filename"</code>	import module
<code>import "filename" as name</code>	import filename as module name
<code>include "filename"</code>	include verbatim text from file
<code>type fnc(type,...);</code>	optional function declaration
<code>type name;</code>	variable declaration
<code>type fnc(type arg,...) {</code> <i>statements</i> <code>return value;</code> <code>}</code>	function definition

## Data types/declarations

boolean (true or false)	<code>bool</code>
tri-state boolean (true, default, or false)	<code>bool3</code>
integer	<code>int</code>
float (double precision)	<code>real</code>
ordered pair (complex number)	<code>pair</code>
character string	<code>string</code>
fixed piecewise cubic Bezier spline	<code>path</code>
unresolved piecewise cubic Bezier spline	<code>guide</code>
color, line type/width/cap, font, fill rule	<code>pen</code>
label with position, alignment, pen attributes	<code>Label</code>
drawing canvas	<code>picture</code>
affine transform	<code>transform</code>
constant (unchanging) value	<code>const</code>
allocate in higher scope	<code>static</code>
no value	<code>void</code>
inhibit implicit argument casting	<code>explicit</code>
structure	<code>struct</code>
create name by data type	<code>typedef type name</code>

## 3D data types (import three;)

ordered triple	<code>triple</code>
3D path	<code>path3</code>
3D guide	<code>guide3</code>
3D affine transform	<code>transform3</code>

## Constants

exponential form	<code>6.02e23</code>
TeX string constant	<code>"abc...de"</code>
TeX strings: special characters	<code>\\, \"</code>
C strings: constant	<code>'abc...de'</code>
C strings: special characters	<code>\\, \" \' \?</code>
C strings: newline, cr, tab, backspace	<code>\n \r \t \b</code>
C strings: octal, hexadecimal bytes	<code>\0-\377 \x0-\xFF</code>

## Operators

arithmetic operations  
modulus (remainder)  
comparisons  
not  
and or (conditional evaluation of RHS)  
and or xor  
cast expression to type  
increment decrement prefix operators  
assignment operators  
conditional expression  
structure member operator  
expression evaluation separator

## Flow control

statement terminator  
block delimiters  
comment delimiters  
comment to end of line delimiter  
exit from while/do/for  
next iteration of while/do/for  
return value from function  
terminate execution  
abort execution with error message

### Flow constructions (if/while/for/do)

<code>if(expr) statement</code> <code>else if(expr) statement</code> <code>else statement</code>
<code>while(expr)</code> <code>statement</code>
<code>for(expr1; expr2; expr3)</code> <code>statement</code>
<code>for(type var : array)</code> <code>statement</code>
<code>do statement</code> <code>while(expr);</code>

`+ - * /`  
`%`  
`== != > >= < <=`  
`!`  
`&& ||`  
`& | ^`  
`(type) expr`  
`++ --`  
`+= -= *= /= %=`  
`expr1 ? expr2 : expr3`  
`name.member`  
`,`

`;`  
`{ }`  
`/* */`  
`//`  
`break;`  
`continue;`  
`return expr;`  
`exit();`  
`abort(string);`

## Arrays

array	<code>type[] name</code>
array element <code>i</code>	<code>name[i]</code>
anonymous array	<code>new type[dim]</code>
array containing <code>n</code> deep copies of <code>x</code>	<code>array(n,x)</code>
length	<code>name.length</code>
cyclic flag	<code>name.cyclic</code>
pop element <code>x</code>	<code>name.pop()</code>
push element <code>x</code>	<code>name.push(x)</code>
append array <code>a</code>	<code>name.append(a)</code>
insert rest arguments at index <code>i</code>	<code>name.insert(i,...)</code>
delete element at index <code>i</code>	<code>name.delete(i)</code>
delete elements with indices in <code>[i,j]</code>	<code>name.delete(i,j)</code>
delete all elements	<code>name.delete()</code>
test whether element <code>n</code> is initialized	<code>name.initialized(n)</code>
array of indices of initialized elements	<code>name.keys</code>
complement of int array in <code>{0,...,n-1}</code>	<code>complement(a,n)</code>
deep copy of array <code>a</code>	<code>copy(a)</code>
array <code>{0,1,...,n-1}</code>	<code>sequence(n)</code>
array <code>{n,n+1,...,m}</code>	<code>sequence(n,m)</code>
array <code>{n-1,n-2,...,0}</code>	<code>reverse(n)</code>
array <code>{f(0),f(1),...,f(n-1)}</code>	<code>sequence(f,n)</code>
array obtained by applying <code>f</code> to array <code>a</code>	<code>map(f,a)</code>
uniform partition of <code>[a,b]</code> into <code>n</code> intervals	<code>uniform(a,b,n)</code>
concat specified 1D arrays	<code>concat(a,b,...)</code>
return sorted array	<code>sort(a)</code>
return array sorted using ordering <code>less</code>	<code>sort(a,less)</code>
search sorted array <code>a</code> for key	<code>search(a,key)</code>
index of first true value of bool array <code>a</code>	<code>find(a)</code>
index of <code>nth</code> true value of bool array <code>a</code>	<code>find(a,n)</code>

## Initialization

initialize variable	<code>type name=value;</code>
initialize array	<code>type[] name={...};</code>

## path connectors

straight segment	<code>--</code>
Beziér segment with implicit control points	<code>..</code>
Beziér segment with explicit control points	<code>..controls c0 and c1.</code>
concatenate	<code>&amp;</code>
lift pen	<code>^^</code>
..tension atleast 1..	<code>::</code>
..tension atleast infinity..	<code>---</code>

## Labels

implicit cast of string <code>s</code> to Label	<code>s</code>
Label <code>s</code> with relative position and alignment	<code>Label(s,real,pair)</code>
Label <code>s</code> with absolute position and alignment	<code>Label(s,pair,pair)</code>
Label <code>s</code> with specified pen	<code>Label(s,pen)</code>

## draw commands

draw path with current pen	<code>draw(path)</code>
draw path with pen	<code>draw(path,pen)</code>
draw labeled path	<code>draw(Label,path)</code>
draw arrow with pen	<code>draw(path,pen,Arrow)</code>
draw path on picture	<code>draw(picture,path)</code>

## fill commands

fill path with current pen	<code>fill(path)</code>
fill path with pen	<code>fill(path,pen)</code>
fill path on picture	<code>fill(picture,path)</code>

## label commands

label a pair with optional alignment <code>z</code>	<code>label(Label,pair,z)</code>
label a path with optional alignment <code>z</code>	<code>label(Label,path,z)</code>
add label to picture	<code>label(picture,Label)</code>

## clip commands

clip to path	<code>clip(path)</code>
clip to path with fill rule	<code>clip(path,pen)</code>
clip picture to path	<code>clip(picture,path)</code>

## pens

Grayscale pen from value in <code>[0,1]</code>	<code>gray(g)</code>
RGB pen from values in <code>[0,1]</code>	<code>rgb(r,g,b)</code>
CMYK pen from values in <code>[0,1]</code>	<code>cmyk(r,g,b)</code>
RGB pen from hexadecimal string	<code>rgb(string)</code>
heximdecimal string from rgb pen	<code>hex(pen)</code>
hsv pen from values in <code>[0,1]</code>	<code>hsv(h,s,v)</code>
invisible pen	<code>invisible</code>
default pen	<code>defaultpen</code>
current pen	<code>currentpen</code>
solid pen	<code>solid</code>
dotted pen	<code>dotted</code>
wide dotted current pen	<code>Dotted</code>
wide dotted pen	<code>Dotted(pen)</code>
dashed pen	<code>dashed</code>
long dashed pen	<code>longdashed</code>
dash dotted pen	<code>dashdotted</code>
long dash dotted pen	<code>longdashdotted</code>
PostScript butt line cap	<code>squarecap</code>
PostScript round line cap	<code>roundcap</code>
PostScript projecting square line cap	<code>extendcap</code>
miter join	<code>miterjoin</code>
round join	<code>roundjoin</code>
bevel join	<code>beveljoin</code>
pen with miter limit	<code>miterlimit(real)</code>
zero-winding fill rule	<code>zerowinding</code>
even-odd fill rule	<code>evenodd</code>
align to character bounding box (default)	<code>nobasealign</code>
align to T <sub>E</sub> X baseline	<code>basealign</code>
pen with font size (pt)	<code>fontsize(real)</code>
LaTeX pen from encoding,family,series,shape	<code>font(strings)</code>
T <sub>E</sub> X pen	<code>font(string)</code>
scaled T <sub>E</sub> X pen	<code>font(string,real)</code>
PostScript font from strings	<code>Courier(series,shape)</code>
pen with opacity in <code>[0,1]</code>	<code>opacity(real)</code>
construct pen nib from polygonal path	<code>makepen(path)</code>
pen mixing operator	<code>+</code>

## path operations

number of segments in path **p**  
number of nodes in path **p**  
is path **p** cyclic?  
is segment **i** of path **p** straight?  
is path **p** straight?  
coordinates of path **p** at time **t**  
direction of path **p** at time **t**  
direction of path **p** at **length(p)**  
unit(**dir(p)**+**dir(q)**)  
acceleration of path **p** at time **t**  
radius of curvature of path **p** at time **t**  
precontrol point of path **p** at time **t**  
postcontrol point of path **p** at time **t**  
arclength of path **p**  
time at which **arclength(p)=L**  
point on path **p** at arclength **L**  
first value **t** at which **dir(p,t)=z**  
time **t** at relative fraction **l** of **arclength(p)**  
point at relative fraction **l** of **arclength(p)**  
point midway along arclength of **p**  
path running backwards along **p**  
subpath of **p** between times **a** and **b**  
times for one intersection of paths **p** and **q**  
times at which **p** reaches minimal extents  
times at which **p** reaches maximal extents  
intersection times of paths **p** and **q**  
intersection times of paths **p** and **a--b**  
intersection times of path **p** crossing  $x = x$   
intersection times of path **p** crossing  $y = z.y$   
intersection point of paths **p** and **q**  
intersection points of **p** and **q**  
intersection of extension of **P--Q** and **p--q**  
lower left point of bounding box of path **p**  
upper right point of bounding box of path **p**  
subpaths of **p** split by  $n$ th cut of **knife**  
winding number of path **p** about pair **z**  
pair **z** lies within path **p**?  
pair **z** lies within or on path **p**?  
path surrounding region bounded by paths  
path filled by **draw(g,p)**  
unit square with lower-left vertex at origin  
unit circle centered at origin  
circle of radius **r** about **c**  
arc of radius **r** about **c** from angle **a** to **b**  
unit **n**-sided polygon  
unit **n**-point cyclic cross

## pictures

add picture **pic** to currentpicture  
add picture **pic** about pair **z**

**length(p)**  
**size(p)**  
**cyclic(p)**  
**straight(p,i)**  
**piecewisestraight(p)**  
**point(p,t)**  
**dir(p,t)**  
**dir(p)**  
**dir(p,q)**  
**accel(p,t)**  
**radius(p,t)**  
**precontrol(p,t)**  
**postcontrol(p,t)**  
**arclength(p)**  
**arctime(p,L)**  
**arcpoint(p,L)**  
**dir(p,z)**  
**reltime(p,l)**  
**relpoint(p,l)**  
**midpoint(p)**  
**reverse(p)**  
**subpath(p,a,b)**  
**intersect(p,q)**  
**mintimes(p)**  
**maxtimes(p)**  
**intersections(p,q)**  
**intersections(p,a,b)**  
**times(p,x)**  
**times(p,z)**  
**intersectionpoint(p,q)**  
**intersectionpoints(p,q)**  
**extension(P,Q,p,q)**  
**min(p)**  
**max(p)**  
**cut(p,knife,n)**  
**windingnumber(p,z)**  
**interior(p,z)**  
**inside(p,z)**  
**buildcycle(...)**  
**strokepath(g,p)**  
**unitsquare**  
**unitcircle**  
**circle(c,r)**  
**arc(c,r,a,b)**  
**polygon(n)**  
**cross(n)**

**add(pic)**  
**add(pic,z)**

## affine transforms

identity transform  
shift by values  
shift by pair  
scale by **x** in the  $x$  direction  
scale by **y** in the  $y$  direction  
scale by **x** in both directions  
scale by real values **x** and **y**  
map  $(x,y) \rightarrow (x+sy,y)$   
rotate by real **angle** in degrees about pair **z**  
reflect about line from **P--Q**

## string operations

concatenate operator  
string length  
position  $\geq$  **pos** of first occurrence of **t** in **s**  
position  $\leq$  **pos** of last occurrence of **t** in **s**  
string with **t** inserted in **s** at **pos**  
string **s** with **n** characters at **pos** erased  
substring of string **s** of length **n** at **pos**  
string **s** reversed  
string **s** with **before** changed to **after**  
string **s** translated via  $\{\{\text{before}, \text{after}\}, \dots\}$   
format **x** using C-style format string **s**  
casts hexadecimal string to an integer  
casts **x** to string using precision **digits**  
current time formatted by **format**  
time in seconds of string **t** using **format**  
string corresponding to **seconds** using **format**  
split **s** into strings separated by **delimiter**

**identity()**  
**shift(real,real)**  
**shift(pair)**  
**xscale(x)**  
**yscale(y)**  
**scale(x)**  
**scale(x,y)**  
**slant(s)**  
**rotate(angle,z=(0,0))**  
**reflect(P,Q)**  
  
**+**  
**length(string)**  
**find(s,t,pos=0)**  
**rfind(s,t,pos=-1)**  
**insert(s,pos,t)**  
**erase(s,pos,n)**  
**substr(s,pos,n)**  
**reverse(s)**  
**replace(s,before,after)**  
**replace(s,string [] [] table)**  
**format(s,x)**  
**hex(s)**  
**string(x,digits=realDigits)**  
**time(format="%a %b %d %T %Z %Y")**  
**seconds(t,format)**  
**time(seconds,format)**  
**split(s,delimiter="")**