Asymptote Reference Card

Program structure/functions

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

Data types/declarations

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

3D data types (import three;)

ordered triple	`triple`
3D path	`path3`
3D guide	`guide3`
3D affine transform	`transform3`

Constants

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

Operators

arithmetic operations	`+ - * /`
modulus (remainder)	`%`
comparisons	`== != > >= < <=`
not	`!`
and or (conditional evaluation of RHS)	`&& ||`
and or xor	`& | ^`
cast expression to type	`(`*type*`)` *expr*
increment decrement prefix operators	`++ --`
assignment operators	`+= -= *= /= %=`
conditional expression	$expr_1$ `?` $expr_2$ `:` $expr_3$
structure member operator	*name*`.`*member*
expression evaluation separator	`,`

Flow control

statement terminator	`;`
block delimeters	`{ }`
comment delimeters	`/* */`
comment to end of line delimiter	`//`
exit from `while`/`do`/`for`	`break;`
next iteration of `while`/`do`/`for`	`continue;`
return value from function	`return` *expr*`;`
terminate execution	`exit();`
abort execution with error message	`abort(string);`

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

`if(`*expr*`)` *statement*
`else if(`*expr*`)` *statement*
`else` *statement*

`while(`*expr*`)`
statement

`for(`$expr_1$`;` $expr_2$`;` $expr_3$`)`
statement

`for(`*type* `var :` *array*`)`
statement

`do` *statement*
`while(`*expr*`);`

Arrays

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

Initialization

initialize variable	*type name*`=`*value*`;`
initialize array	*type*`[]` *name*`={...};`

path connectors

straight segment	`--`
Beziér segment with implicit control points	`..`
Beziér segment with explicit control points	`..controls c0 and c1..`
concatenate	`&`
lift pen	`^^`
..tension atleast 1..	`::`
..tension atleast infinity..	`---`

Labels

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

draw commands

draw path with current pen	`draw(path)`
draw path with pen	`draw(path,pen)`
draw labeled path	`draw(Label,path)`
draw arrow with pen	`draw(path,pen,Arrow)`
draw path on picture	`draw(picture,path)`

fill commands

fill path with current pen	`fill(path)`
fill path with pen	`fill(path,pen)`
fill path on picture	`fill(picture,path)`

label commands

label a pair with optional alignment z	`label(Label,pair,z)`
label a path with optional alignment z	`label(Label,path,z)`
add label to picture	`label(picture,Label)`

clip commands

clip to path	`clip(path)`
clip to path with fill rule	`clip(path,pen)`
clip picture to path	`clip(picture,path)`

pens

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

path operations

number of segments in path `p`	`length(p)`
number of nodes in path `p`	`size(p)`
is path `p` cyclic?	`cyclic(p)`
is segment `i` of path `p` straight?	`straight(p,i)`
is path `p` straight?	`piecewisestraight(p)`
coordinates of path `p` at time `t`	`point(p,t)`
direction of path `p` at time `t`	`dir(p,t)`
direction of path `p` at `length(p)`	`dir(p)`
unit(dir(p)+dir(q))	`dir(p,q)`
acceleration of path `p` at time `t`	`accel(p,t)`
radius of curvature of path `p` at time `t`	`radius(p,t)`
precontrol point of path `p` at time `t`	`precontrol(p,t)`
postcontrol point of path `p` at time `t`	`postcontrol(p,t)`
arclength of path `p`	`arclength(p)`
time at which `arclength(p)=L`	`arctime(p,L)`
point on path `p` at arclength `L`	`arcpoint(p,L)`
first value `t` at which `dir(p,t)=z`	`dir(p,z)`
time `t` at relative fraction `l` of `arclength(p)`	`reltime(p,l)`
point at relative fraction `l` of `arclength(p)`	`relpoint(p,l)`
point midway along arclength of `p`	`midpoint(p)`
path running backwards along `p`	`reverse(p)`
subpath of `p` between times `a` and `b`	`subpath(p,a,b)`
times for one intersection of paths `p` and `q`	`intersect(p,q)`
times at which `p` reaches minimal extents	`mintimes(p)`
times at which `p` reaches maximal extents	`maxtimes(p)`
intersection times of paths `p` and `q`	`intersections(p,q)`
intersection times of paths `p` and `a--b`	`intersections(p,a,b)`
intersection times of path `p` crossing $x =$`x`	`times(p,x)`
intersection times of path `p` crossing $y =$`z.y`	`times(p,z)`
intersection point of paths `p` and `q`	`intersectionpoint(p,q)`
intersection points of `p` and `q`	`intersectionpoints(p,q)`
intersection of extension of `P--Q` and `p--q`	`extension(P,Q,p,q)`
lower left point of bounding box of path `p`	`min(p)`
upper right point of bounding box of path `p`	`max(p)`
subpaths of `p` split by nth cut of `knife`	`cut(p,knife,n)`
winding number of path `p` about pair `z`	`windingnumber(p,z)`
pair `z` lies within path `p`?	`interior(p,z)`
pair `z` lies within or on path `p`?	`inside(p,z)`
path surrounding region bounded by paths	`buildcycle(...)`
path filled by `draw(g,p)`	`strokepath(g,p)`
unit square with lower-left vertex at origin	`unitsquare`
unit circle centered at origin	`unitcircle`
circle of radius `r` about `c`	`circle(c,r)`
arc of radius `r` about `c` from angle `a` to `b`	`arc(c,r,a,b)`
unit `n`-sided polygon	`polygon(n)`
unit `n`-point cyclic cross	`cross(n)`

pictures

add picture `pic` to currentpicture	`add(pic)`
add picture `pic` about pair `z`	`add(pic,z)`

affine transforms

identity transform	`identity()`
shift by values	`shift(real,real)`
shift by pair	`shift(pair)`
scale by `x` in the x direction	`xscale(x)`
scale by `y` in the y direction	`yscale(y)`
scale by `x` in both directions	`scale(x)`
scale by real values `x` and `y`	`scale(x,y)`
map $(x, y) \to (x+$`s`$y, y)$	`slant(s)`
rotate by real `angle` in degrees about pair `z`	`rotate(angle,z=(0,0))`
reflect about line from `P--Q`	`reflect(P,Q)`

string operations

concatenate operator	`+`
string length	`length(string)`
position $\geq$ `pos` of first occurence of `t` in `s`	`find(s,t,pos=0)`
position $\leq$ `pos` of last occurence of `t` in `s`	`rfind(s,t,pos=-1)`
string with `t` inserted in `s` at `pos`	`insert(s,pos,t)`
string `s` with `n` characters at `pos` erased	`erase(s,pos,n)`
substring of string `s` of length `n` at `pos`	`substr(s,pos,n)`
string `s` reversed	`reverse(s)`
string `s` with `before` changed to `after`	`replace(s,before,after)`
string `s` translated via `{{before,after},...}`	`replace(s,string [][] table)`
format `x` using C-style format string `s`	`format(s,x)`
casts hexidecimal string to an integer	`hex(s)`
casts `x` to string using precision `digits`	`string(x,digits=realDigits)`
current time formatted by `format`	`time(format="%a %b %d %T %Z %Y")`
time in seconds of string `t` using `format`	`seconds(t,format)`
string corresponding to `seconds` using `format`	`time(seconds,format)`
split `s` into strings separated by `delimiter`	`split(s,delimiter="")`