Cocos2 and pyglet Quick Reference (v1.4; © Richard Jones, richard@mechanicalcat.net 2010)

from cocos.director import director — by default cocos2d will auto-scale requested dimensions to the window size

`director.init(`*arguments*`)`	args include all pyglet.window.Window args and do_not_scale*
`director.run(`*scene*`)`	run the Scene
`director.replace(`*scene*`)`	replace the currently-running Scene with the supplied one
`director.push(`*scene*`)`	run the supplied Scene with the ability to return to the current one
`director.pop()`	return to the previous scene after a `push()`
`director.get_window_size()`	return the (width, height) of the window
`director.get_virtual_coordinates(x, y)`	map window coordinates to logical scene coordinates**
`director.scene`	the currently-active Scene
`director.return_value`	the value from the last Scene `end()`
`director.window`	the pyglet.window.Window

* turns auto-scaling off
** if auto-scaling is on

A cocosnode is a layer, sprite, text, canvas, scene, ... (so sprite.position, layer.scale = .5, scene.pause(), ...)

`.add(`*node*`), .remove(`*node*`), .kill()`	... scene.add(layer), sprite.add(text), layer.remove(sprite) **or** sprite.kill()
`.schedule(`*callback*`)`	... and unschedule() etc. as per pyglet.clock API
`.pause(), .resume()`	pause and resume the execution of actions & scheduled calls
`.are_actions_running()`	determine whether any actions are running
`.on_enter(), .on_exit()`	called as the node enters and leaves the stage (don't forget to `super()`)
`.anchor`	pixel that position is relative to; used to transform about*
`.position`	position in (x,y) coordinates
`.scale`	scale where 1.0 the default value
`.rotation`	rotation in degrees

* default is center

from cocos.scene import Scene

`scene.end(return_value)`	end the Scene and set `director.return_value`
`class` `MyScene(`**`cocos.layer.Scene`**`):`	
`def __init__(self):`	initialise at creation; don't forget to `super(MyScene, self).__init__()`

from cocos.layer import ...

`Layer()`	standard layer containing sprites, text, layers, ...
`MultiplexLayer(`*layer, layer, ...*`)`	composite layer that displays one layer of many at a time
`PythonInterpreterLayer()`	runs an interactive Python interpreter in a Layer
`ScrollableLayer(parallax=1)`	requires the parent to be cocos.layers.ScrollingManager
`ColorLayer(r, g, b, a)`	solid color layer
`class` `MyLayer(`**`cocos.layer.Layer`**`):`	
`def __init__(self):`	initialise at creation; don't forget to `super(MyLayer, self).__init__()`
`is_event_handler = False`	True to register standard pyglet event handlers on this layer
	cocos2d adds scaling-aware `on_cocos_resize(`*width, height*`)`

`from cocos.sprite import Sprite`	`sprite = Sprite('data/ship.png')`
`opacity`	opacity of the sprite where 0 is transparent and 255 is solid
`color`	color in R,G,B format where 0,0,0 is black and 255,255,255 is white

from cocos.text import ...

`Label(text, position, ...)`	plain-text label
`HTMLLabel(text, position, ...)`	HTML 4.01 subset text label (see pyglet.text.formats.html for details)
`RichLabel(text, position, ...)`	rich text label with markup as per pyglet.text.DocumentLabel

```
import cocos.tiles
level = cocos.tiles.load('my-level.xml')
map = level['map-1']
scene = cocos.scene.Scene(map)
map.set_view(x, y, width, height)
note: use do_not_scale when using tile maps
```

scrolling...

```
manager = cocos.layers.ScrollingManager()
manager.add(map)
scene = cocos.scene.Scene(manager)
manager.set_focus(x, y)   or...
manager.force_focus(x, y)
```

keboard status handler and key constants

```
from pyglet.window import key
keys = key.KeyStateHandler()
director.window.push_handlers(keys)
```

`key.RIGHT`	`key.SPACE`
`key.LEFT`	`key.A -> key.Z`
`key.UP`	`key._0 -> key._9`
`key.DOWN`	`key.ENTER`

`from cocos.menu import Menu, MenuItem` also EntryMenuItem, ToggleMenuItem, ImageMenuItem, ...

```
menu = cocos.menu.Menu('My Game Title')
menu.create_menu(([
      MenuItem('Play', lambda: director.push(TheGameScene())),
      MenuItem('Quit', pyglet.app.exit)])
menu.on_quit = pyglet.app.exit
director.run(cocos.scene.Scene(menu))
```

Cocos2 and pyglet Quick Reference (v1.4; © Richard Jones, richard@mechanicalcat.net 2010)

	from cocos.actions import ...	cocosnode.do(action) (sprite.do, layer.do, ...)
Translation	MoveBy(delta, duration=5)	Moves the sprite *delta*=(x, y) pixels
	MoveTo	Moves the sprite to *position*=(x,y)
	JumpBy(delta, height=100, jumps=1, duration=5)	Jump the sprite *delta*=(x, y), *height* pixels using *jumps* hops
	JumpTo(position, height=100, jumps=1, duration=5)	Jump the sprite to *position*=(x,y), *height* pixels using *jumps* hops
	Bezier(bezier, duration=5)	Move the sprite through the *bezier* curve (cocos.path.Bezier instance)
	Place(position)	Instantly place the sprite at the *position*=(x, y)
Transform	ScaleBy(scale, duration=5)	Scale the sprite by *scale* times
	ScaleTo(scale, duration=5)	Scale the sprite to *scale*
	RotateBy(angle, duration=5)	Rotate the target by *angle* degrees
	RotateTo(angle, duration=5)	Rotate the sprite to the given *angle*
Visibility	Show()	Show the sprite
	Hide()	Hide the sprite from view
	Blink(blinks, duration)	Blink the sprite the number of *blinks* over the *duration* seconds
	ToggleVisibility()	Show if hidden and hide if shown
	FadeIn(duration)	Fade the sprite into view over *duration* seconds
	FadeOut(duration)	Fade the sprite out of view over *duration* seconds
	FadeTo(opacity, duration)	Fade the sprite to a specific *opacity* over *duration* seconds
Modifiers	Accelerate(action, rate=2)	Accelerate the *action* at its end by the given *rate* (1 is linear)
	AccelDeccel(action)	Accelerate the *action* in its middle
	Speed(action, rate)	Speed up or slow down the *action* by the given *rate* (1 is normal)
	Reverse(action)	Perform the *action* in reverse
Combine	Sequence(action, action) (+ operator)	Execute actions in sequence
	Spawn(action, action) (| operator)	Execute actions at the same time
	Repeat(action)	Repeat an action (or composite set of actions) forever
	Loop(action, times) (* operator)	Loop the action *n* times
Special	Delay(time)	Delay for *time* seconds
	RandomDelay(low, high)	Delay for some seconds below *low* and *high*
	CallFunc(callable)	Invoke the *callable* (with no arguments)
	CallFuncS(callable)	Invoke the *callable* with the sprite as the first argument
	OrbitCamera(*spherical coordinate arguments*)	Orbits the camera around the center of the screen
Move	Move()	Move the sprite based on sprite parameters (velocity, acceleration, ...)
	BoundedMove(width, height)	As above but limit movement to 0 < x < width and 0 < y < height
	WrappedMove(width, height)	As above but wrap movement outside 0 < x < width and 0 < y < height
	Driver()	Drive the sprite like a car using sprite parameters (direction, speed, ...)

```
class MyAction(cocos.actions.Action):
      def init(self):                  gets called at initialization time, before a target is defined
      def step(self, dt):              called every frame with dt being the number of seconds since last call
      def done(self):                  return False while the step method must be called
      def start(self):                 start executing an action; self.target is assigned and this method is called
      def stop(self):                  after we finish executing an action this method is called
class MyIntervalAction(cocos.actions.IntervalAction):
      def update(self, t):             called every frame with t ranging from 0..1
      (also init, start and stop)
```

Cocos2 and pyglet Quick Reference (v1.4; © Richard Jones, richard@mechanicalcat.net 2010)

import pyglet

`window = pyglet.window.Window(...)`	create a window with optional arguments
`pyglet.app.run()`	run pyglet's main loop to handle events

registering event handlers

`@window.event`	attach the following function to the window as the handler for the named event
`def on_draw():`	see below for all possible window event names
`  window.clear()`	clear the window to the pyglet.gl.glClearColor color
`  ...`	put your other drawing code here
`window.push_handlers(on_draw)`	alternative method of pushing the on_draw handler to the window
`class MyClass(object):`	
`  def on_draw(self):`	
`  def on_text(self, text):`	return pyglet.event.EVENT_HANDLED (True) if the event has been handled
`my_object = MyClass()`	
`window.push_handlers(my_object)`	push all handlers defined on my_object onto the window's event stack

Window arguments	(there are other arguments, these are just the most common)
`fullscreen`	make the window fullscreen
`width=640, height=480`	create the window with these dimensions (ignored if fullscreen)
`resizable=False`	allow the user to resize the window
`vsync=True`	synchronise to the monitor to avoid flicker
`caption=sys.argv[0]`	set the window title text
`config=None`	a display config as per pyglet.gl.Config
`screen=None`	the screen to use if fullscreen

Image handling

`pyglet.image.load(`*`filename, file=None`*`)`	load the image from the named file or supplied file object
`SolidColorImagePattern`	create an image filled with a single color
`CheckerImagePattern`	create an image with a tileable checker image of two colors
`image.width, image.height`	image dimensions in pixels
`image.anchor_x, image.anchor_y`	coordinate of anchor, relative to bottom-left corner of image
`image.blit(`*`x, y, z=0`*`)`	render the image to the active framebuffer
`image.save(`*`filename, file=None`*`)`	save the image as a PNG file
`image.texture`	a pyglet.image.Texture view of this image
`texture.target, texture.id`	OpenGL texture target and id
`texture.tex_coords`	12-tuple of float texture coordinates (may not be simply 0 and 1)
`get_buffer_manager().get_color_buffer()`	get the active framebuffer as an Image
`load_animation(filename, file=None)`	load an animation from a file - currently only GIF is supported
`Animation.from_image_sequence(sequence, period, loop=False)`	create an animation from a sequence of images

Sprites	(all attributes are re-assignable)
`pyglet.sprite.Sprite(`*`image`*`, ...)`	create a sprite from the image *or* Animation instance
`sprite.position`	position of the sprite in (x, y) (also as sprite.x, sprite.y)
`sprite.image`	image rendered for the sprite (image anchor is honored)
`sprite.rotation`	sprite rotation in degrees
`sprite.scale`	amount to scale the sprite image by - 1.0 is unscaled
`sprite.opacity`	control transparency - 0 is fully transparent, 255 is fully opaque
`sprite.color`	coloring of sprite image, normal (white) is R, G, B (255, 255, 255)
`sprite.visible`	boolean controlling sprite visibility
`sprite.draw()`	render the sprite to the active framebuffer

Text rendering	(see the docs for the complete, extensive set of options you may pass)
`pyglet.text.Label(text, ...)`	lay out some plain text
`pyglet.text.HTMLLabel(text, ...)`	lay out some HTML (4.01, limited) text
`text.draw()`	render the laid-out text

Media playback

`pyglet.media.load(`*`filename`*`, file=None)`	load the media file (audio, video or both) as a "source"
`source.audio_format`	an instance of pyglet.media.AudioFormat or None
`source.video_format`	an instance of pyglet.media.VideoFormat or None
`source.info`	a pyglet.media.SourceInfo giving title, author, etc. if known
`source.play()`	convenience method to immediately play the source
`player = Player()`	create a player to manage playback; see possible events below
`player.queue(source)`	queue the source to be played
`player.play(), .pause(), .stop()`	control playback
`player.time, player.seek(`*`time`*`)`	report current position and seek to a different *time*
`player.get_texture()`	get the current video frame as a pyglet.image.Texture
`player.eos_action`	the action of the player when it reaches the end of the current source

Cocos2 and pyglet Quick Reference (v1.4; © Richard Jones, richard@mechanicalcat.net 2010)

Graphics abstraction

`pyglet.graphics.Batch()`	batch up graphics drawing operations
`pyglet.sprite.Sprite(image, batch=batch)`	create a sprite belonging to the batch
`batch.draw()`	**much** faster than individual sprite.draw() calls
`pyglet.graphics.Group`	group common OpenGL state objects in a batch
`pyglet.graphics.OrderedGroup`	arbitrarily order objects in a batch (typically for display sorting)
`pyglet.graphics.TextureGroup`	enable and bind a texture for a group of objects in a batch
`batch.add(`*`count, mode, group, *data`*`)`	create an OpenGL vertex list in the batch using *data*'s items
`batch.add(`*`count, mode, group,`* `indices, *data)`	create an OpenGL indexed vertex list

```
# draw a white line from (0, 1) to (1, 0)
vertex_list = batch.add(2, GL_LINES, None, ('v2f', (0.0, 1.0, 1.0, 0.0)),
                                           ('c4B', (255, 255, 255, 255) * 2))
```

Resources	abstract storage of application resources in directories or ZIP files
`pyglet.resource.image(`*`filename`*`)`	load the named image file
`pyglet.resource.media(`*`filename`*`)`	load the named media file
`pyglet.resource.add_font(`*`filename`*`)`	make a font available to pyglet's text rendering
`pyglet.resource.file(`*`filename`*`)`	open the named resource file, returning a file object
`pyglet.resource.location(`*`filename`*`)`	return the location of the resource file (only useful for on-disk files)
`pyglet.resource.path`	list containing the places to look for resources
`pyglet.resource.reindex()`	should be called if the path is modified

Clock handling	note: pyglet.app.run() automatically calls pyglet.clock.tick()
`pyglet.clock.schedule(`*`callback`*`)`	*callback* when the clock is ticked, passing the seconds since last call
`...unschedule(`*`callback`*`)`	remove *callback* from the schedule
`...schedule_interval(`*`callback, n`*`)`	*callback* every *n* seconds
`...schedule_once(`*`callback, n`*`)`	*callback* once in *n* seconds
`fps = pyglet.clock.ClockDisplay()`	a simple FPS counter .. use fps.draw() to display

pyglet window events

`on_key_press(`*`symbol, modifiers`*`)`	*symbol* and *modifiers* as in pyglet.window.key keys and MOD_
`on_key_release(`*`symbol, modifiers`*`)`	as above
`on_text(`*`text`*`)`	*text* is a unicode string of the text input
`on_text_motion(`*`motion`*`)`	*motion* as in pyglet.window.key.MOTION_*
`on_text_motion_select(`*`motion`*`)`	as above but during a text selection event (MOD_SHIFT held)

`on_mouse_press(`*`x, y, button, modifiers`*`)`	*buttons* and *modifiers* pressed at position (*x, y*)
`on_mouse_release(`*`x, y, button, modifiers`*`)`	as above but *buttons* released
`on_mouse_motion(`*`x, y, dx, dy`*`)`	mouse absolute (*x, y*) and movement (*dx, dy*) since last event
`on_mouse_drag(`*`x, y, dx, dy, buttons, modifiers`*`)`	as above but with *buttons* and *modifiers* held
`on_mouse_scroll(`*`x, y, dx, dy`*`)`	mouse scroll wheel scrolled by (*dx, dy*) at position (*x, y*)
`on_mouse_enter(`*`x, y`*`)`	mouse entered window
`on_mouse_leave(`*`x, y`*`)`	mouse exited window

`on_resize(`*`width, height`*`)`	the window has been created or resized

`on_draw()`	application should draw (not relevant to cocos2d)
`on_show()`	window has been made visible (or created)
`on_hide()`	window has been hidden
`on_close()`	window close button pressed
`on_expose()`	redraw is needed
`on_move(`*`x, y`*`)`	window has been moved to position (*x, y*)
`on_activate()`	window has been activated (focused)
`on_deactivate()`	window has been deactivated (lost focus)
`on_context_lost()`	window's OpenGL context was lost (no drawing possible)
`on_context_state_lost()`	window's OpenGL context state was destroyed by pyglet

pyglet media player events

`on_player_eos()`	the player has run out of sources
`on_source_group_eos()`	the current source group has run out of data
`on_eos()`	the current source has run out of data

pyglet sprite event

`on_animation_end`	the sprite's image animation has ended