Pytransitions Quick Reference Sheet

Note: For quick reference only. Mostly code demo. Full guide w/ more explanations found here: https://github.com/pytransitions/transitions

Quick setup

```
class Matter(object):
    pass
lump = Matter()

states=['solid', 'liquid', 'gas', 'plasma']
transitions = [ { 'trigger': 'melt', 'source': 'solid', 'dest': 'liquid' } ]

machine = Machine(lump, states=states, transitions=transitions, initial='liquid')

lump.melt()
lump.triger('melt')
```

States

Callbacks
***on_enter**: callbacks declared on the destination state*
***on_exit**: callbacks declared on the source state)*
	`class Matter(object):`
	`    def say_hello(self): print("hello, new state!")`
	`    def say_goodbye(self): print("goodbye, old state!")`
	
	`states = [`
	`    State(name='solid', on_exit=['say_goodbye']),`
	`    State('liquid')`
	`    'gas',`
	`    { 'name': 'plasma'}`
	`    ]`
	
	`machine.on_enter_gas('say_hello')`

Check state	`lump.state`
	`lump.is_gas()`
	`machine.get_state(lump.state).name`

Transitions

Add trans	transitions = [{ 'trigger': 'melt', 'source': 'solid', 'dest': 'liquid' }, { 'trigger': 'evaporate', 'source': 'liquid', 'dest': 'gas' }] transitions = [['melt', 'solid', 'liquid'], ['evaporate', 'liquid', 'gas']] machine.**add_transition('melt', source='solid', dest='liquid')**
Hide trigger error	m = Machine(lump, states, initial='solid', **ignore_invalid_triggers=True**)
Get its triggers	m.**get_triggers('solid')** m.**get_triggers('solid', 'liquid', 'gas', 'plasma')**
Forced trans	lump.**to_liquid()**

Triggers from many states	machine.add_transition('transmogrify', **['solid', 'liquid', 'gas']**, 'plasma')
	machine.add_transition('to_liquid', **'*'**, 'liquid')

Reflexive trans	machine.add_transition('touch', ['liquid', 'gas', 'plasma'], **'=', after='change_shape'**)

Ordered trans	states = ['A', 'B', 'C']
	machine = Machine(states=states, initial='A')
	machine.add_ordered_transitions()
	machine.next_state()

Queued trans	machine = Machine(states=states, **queued=True**)

Conditional trans	class Matter(object):
	def is_flammable(self): return False
	def is_really_hot(self): return True
	machine.add_transition('heat', 'solid', 'gas', **conditions='is_flammable'**)
	if is_flammable true, 'solid' -> 'gas'; otherwise, if is_really_hot true, 'solid' -> 'liquid'

	machine.add_transition('heat', 'solid', 'liquid', **conditions='is_really_hot'**)	
	machine.add_transition('heat', 'solid', 'gas', **unless=['is_flammable', 'is_really_hot']**)	*run heat() unless both is_flammable and is_really_hot return False*
	lump.heat(**temp=74**) # = lump.trigger('heat', temp=74)	*passed to is_flamable as optional kwarg*

Callbacks	***before, after***
	(before / after transition)
		class Matter(object):
		def make_noises(self): print("HISSSSSSSSSSSSSSSS")
		def disappear(self): print("where'd all the liquid go?")
		
		transitions = [
		{'trigger': 'melt', 'source': 'solid', 'dest': 'liquid', **'before': 'make_noises'**},
		{'trigger': 'evaporate', 'source': 'liquid', 'dest': 'gas', **'after': 'disappear'** }]

	prepare
	(executed as soon as trans starts, before conditions or callbacks)
		class Matter(object):
		heat = False
		attempts = 0
		def heat_up(self): self.heat = random.random() < 0.25
		def count_attempts(self): self.attempts += 1
		def is_really_hot(self): return self.heat
		def stats(self): print('#Attempts: %i' %self.attempts)
		
		transitions = [{ 'trigger': 'melt', 'source': 'solid', 'dest': 'liquid', **'prepare': ['heat_up', 'count_attempts']**, 'conditions': 'is_really_hot', 'after': 'stats'}]

	default callbacks *(declared on Machine, before / after trans)*	m = Machine(lump, states, **before_state_change='make_noises', after_state_change='disappear'**)

	prepare_event
	(executed once before trans are processed)
	finalize_event
	(executed even if no trans occur or exception is raised)
	send_event
	(error is retrievable)
		class Matter(object):
		def prepare(self, event): print("I am ready!")
		def raise_error(self, event): raise ValueError("Oh no")
		def finalize(self, event): print("Result: ", type(event.error), event.error)
		
		m = Machine(lump, states, **prepare_event='prepare'**, before_state_change='raise_error', **finalize_event='finalize', send_event=True**)

Alternative initialization patterns

Model inherited from Machine class

```
class Matter(Machine):
    def __init__(self):
        states = ['solid', 'liquid', 'gas']
        Machine.__init__(self, states=states, initial='solid')
        self.add_transition('melt', 'solid', 'liquid')
```

Add/remove multiple models in single machine

```
lump1 = Matter()
lump2 = Matter()

machine = Machine(states=states, transitions=transitions, initial='solid', add_self=False)

machine.add_model(lump1)
machine.add_model(lump2, initial='liquid')

machine.remove_model([lump1, lump2])
del lump1
del lump2
```