Argparse concept	Example	Will be covered in argparse2cwl
prefix chars	parser.add_argument('+f') parser.add_argument('++bar')	yes

parents	>>> parent_parser = argparse.ArgumentParser(add_help=**False**)
	>>> parent_parser.add_argument('--parent', type=int)
	
	>>> foo_parser = argparse.ArgumentParser(parents=[parent_parser])
	>>> foo_parser.add_argument('foo')
	>>> foo_parser.parse_args(['--parent', '2', 'XXX'])
	Namespace(foo='XXX', parent=2)
		yes

argument default	Generally, argument defaults are specified either by passing a default to add_argument() or by calling the set_defaults() methods with a specific set of name-value pairs. Sometimes however, it may be useful to specify a single parser-wide default for arguments. This can be accomplished by passing the argument_default= keyword argument to ArgumentParser. For example, to globally suppress attribute creation on parse_args() calls, we supply argument_default=SUPPRESS:
	>>>
	>>> parser = argparse.ArgumentParser(argument_default=argparse.SUPPRESS)
	>>> parser.add_argument('--foo')
	>>> parser.add_argument('bar', nargs='?')
	>>> parser.parse_args(['--foo', '1', 'BAR'])
	Namespace(bar='BAR', foo='1')
	>>> parser.parse_args([])
	Namespace()
		No

Actions:	'append_const' - This stores a list, and appends the value	Yes

append_const	specified by the const keyword argument to the list. (Note that the constkeyword argument defaults to None.) The 'append_const' action is typically useful when multiple arguments need to store constants to the same list. For example:
	>>>
	>>> parser = argparse.ArgumentParser()
	>>>parser.add_argument('--str', dest='types', action='append_const', const=str)
	>>>parser.add_argument('--int', dest='types', action='append_const', const=int)
	>>> parser.parse_args('--str --int'.split())
	Namespace(types=[<class 'str'>, <class 'int'>])	

count	'count' - This counts the number of times a keyword argument occurs. For example, this is useful for increasing verbosity levels:
	>>>
	>>> parser = argparse.ArgumentParser()
	>>>parser.add_argument('--verbose', '-v', action='count')
	>>> parser.parse_args(['-vvv'])
	Namespace(verbose=3)
		neccessary?

custom actions	>>> class FooAction(argparse.Action):
	... def __init__(self, option_strings, dest, nargs=None, **kwargs):
	... if nargs is not None:
	... raise ValueError("nargs not allowed")
	... super(FooAction, self).__init__(option_strings, dest, **kwargs)
	... def __call__(self, parser, namespace, values, option_string=None):
	... print('%r %r %r' % (namespace, values, option_string))
	... setattr(namespace, self.dest, values)
	...
	>>> parser = argparse.ArgumentParser()
	>>> parser.add_argument('--foo', action=FooAction)
	>>> parser.add_argument('bar', action=FooAction)
	>>> args = parser.parse_args('1 --foo 2'.split())
		No

	Namespace(bar=None, foo=None) '1' None Namespace(bar='1', foo=None) '2' '--foo' >>> args Namespace(bar='1', foo='2')	
custom types		No (CWL doesn't support)

nargs = N	>>> parser = argparse.ArgumentParser()
	>>> parser.add_argument('--foo', nargs=2)
	>>> parser.add_argument('bar', nargs=1)
	>>> parser.parse_args('c --foo a b'.split())
	Namespace(bar=['c'], foo=['a', 'b'])
		No (CWL doesn't support)

nargs = '?' - const keyword
 	
store_const
	Note that for optional arguments, there is an additional case - the option string is present but not followed by a command-line argument. In this case the value from const will be produced. Some examples to illustrate this:
	>>>
	>>> parser = argparse.ArgumentParser()
	>>> parser.add_argument('--foo', nargs='?', const='c', default='d')
	>>> parser.add_argument('bar', nargs='?', default='d')
	>>> parser.parse_args(['XX', '--foo', 'YY'])
	Namespace(bar='XX', foo='YY')
	>>> parser.parse_args(['XX', '--foo'])
	Namespace(bar='XX', foo='c')
	>>> parser.parse_args([])
	Namespace(bar='d', foo='d')
	_____
	
	'store_const' - This stores the value specified by the const keyword argument. The 'store_const' action is most commonly used with optional arguments that specify some sort of flag. For example:
	>>>
	>>> parser = argparse.ArgumentParser()
	>>> parser.add_argument('--foo', action='store_const', const=42)
	>>> parser.parse_args(['--foo'])
	Namespace(foo=42)
		No, const better to be specified in job.json

mutual exclusive	>>> parser = argparse.ArgumentParser(prog='PROG')
	>>> group = parser.add_mutually_exclusive_group()
	>>> group.add_argument('--foo', action='store_true')
	>>> group.add_argument('--bar', action='store_false')
	>>> parser.parse_args(['--foo'])
	Namespace(bar=True, foo=True)
	>>> parser.parse_args(['--bar'])
	Namespace(bar=False, foo=False)
	>>> parser.parse_args(['--foo', '--bar'])
	usage: PROG [-h] [--foo | --bar]
	PROG: error: argument --bar: not allowed with argument --foo
		No (CWL doesn't support)

set defaults	>>> parser = argparse.ArgumentParser()
	>>> parser.add_argument('foo', type=int)
	>>> parser.set_defaults(bar=42, baz='badger')
	>>> parser.parse_args(['736'])
	Namespace(bar=42, baz='badger', foo=736)
		Yes

nargs=argparse.REMAINDER	argparse.REMAINDER. All the remaining command-line arguments are gathered into a list. This is commonly useful for command line utilities that dispatch to other command line utilities:
	>>>
	>>> parser = argparse.ArgumentParser(prog='PROG')
	>>> parser.add_argument('--foo')
	>>> parser.add_argument('command')
	>>> parser.add_argument('args', nargs=argparse.REMAINDER)
	>>> print(parser.parse_args('--foo B cmd --arg1 XX ZZ'.split()))
	Namespace(args=['--arg1', 'XX', 'ZZ'], command='cmd', foo='B')
		neccessary?

parse known args	>>> parser = argparse.ArgumentParser()
	>>> parser.add_argument('--foo', action='store_true')
	>>> parser.add_argument('bar')
	>>> parser.parse_known_args(['--foo', '--badger', 'BAR', 'spam'])
	(Namespace(bar='BAR', foo=True), ['--badger', 'spam'])
		Yes