List comprehension

```
res = [func(i) for i in some_iter if func2(i)]
res = [i ** 0.5 for i in [-1, 0 , 1, 2 ,3] if i >= 0]
res = (func(i) for i in some_iter if func2(i))
res = {func(i) for i in some_iter if func2(i)}
```

inline if

```
res = x if x >= 0 else -x
# res = (x >= 0 ? x : -x)
```

DRY

Потоки

```
import threading

th = threading.Thread(None, func, None, args, kwargs)
th.daemon = True

th.start()

th.is_alive()
th.join(timeout)
```

Потоки

- threading.enumerate
- threading.local

```
mydata = threading.local()
mydata.x = 1
```

- sys.setcheckinterval(N)

Потоки

- winpdb
- GIL (cache, ...)
- Не отменяемые (thread2)

```python
class MyTask(object):
    def my_thread_func(self):
        pass

    def start_thread(self):
        self.th = Thread(None, self.my_thread_func)
```

Примитивы синхронизации

- threading.Lock
- threading.Semaphore
- threading.RLock
- threading.Event
- threading.Condition

```
lock = threading.Lock()
...
# lock.acquire()
with lock:
    pass
# lock.release()
```

```python
cvar = threading.Condition()
...
def th1():
    with cvar:
        cvar.wait()
        ...

def th2():
    with cvar:
        cvar.wait()
        ...

def th3():
    with cvar:
        #cvar.notify()
        cvar.notify_all()
```

Queue

```python
import Queue

q = Queue.Queue(maxsize=0)
q.put(val, block=True, timeout=None)
q.get(block=True, timeout=None)
```

concurrent - python 3.2

```
def worker(param_q, result_q, func):
    while True:
        param = param_q.get()
        if param is None:
            break
        result_q.put((param, func(param)))

result_q = Queue.Queue()
param_q = Queue.Queue()
workers = []
worker_params = (param_q, result_q, func)

for i in range(pool_sz):
    th = threading.Thread(None, worker,
                          "worker-{}".format(i),
                          worker_params)
    th.daemon = True
    th.start()
```

```python
        workers.append(th)

    # params_q.put(...)
    # result_q.get(...)

    for i in range(pool_sz):
        params_q.put(None)

    for th in workers:
        th.join()
```

concurrent - python 3.2

```
from concurrent.futures import ThreadPoolExecutor
with ThreadPoolExecutor(max_workers=4) as pool:
    res = pool.map(func, iter)
    future = pool.submit(func, ....)

    #future.cancel()
    #future.done()
    print future.result(timeout=None)
```

multiprocessing