

```
1 stack.insert(0, val)
2 stack.pop(0)
```

```
1 def load_file(fname):  
2     lines = []  
3     with open(fname) as fd:  
4         for line in fd:  
5             # ....  
6             lines.append(processed_line)  
7     return lines
```

```
1 def load_file2(fname):  
2     with open(fname) as fd:  
3         for line in fd:  
4             # ....  
5             yield processed_line
```

```
1 def load_file3 (fd):
2     for line in fd:
3         # ....
4         yield processed_line
5
6 def execute (fname):
7     for cmd in load_file3 (fname):
8         # .....
```

```

1  class Forth( object ):
2      stack = []
3
4
5  class Forth( object ):
6      def __init__( self ):
7          self._stack = []
8
9
10 class Forth( object ):
11     def __init__( self , statements ):
12         self.current_parameter = None
13
14     def execute( self ):
15         for .....:
16             # .....
17             self.current_parameter = some_data
18             run_command

```

```
1 a, b = self.stack.pop(), self.stack.pop()
2 a[func1()] = func2()
3
4
5 cmd = s.split()[0]
6 param = s.split()[1]
```

```
1
2 class Forth(object):
3     # ....
4     def run(self):
5         for s in self._statements:
6             if hasattr(self, s.split()[0]):
7                 method = getattr(self, s.split()[0])
8                 method()
```

## LEAP vs EAFP

```
1 def add_LEAP( stack ):
2     if len( stack ) < 2:
3         raise SomeNiceClass()
4     stack.append( stack.pop() + stack.pop() )
5
6 def add_EAFP( stack ):
7     stack.append( stack.pop() + stack.pop() )
```



## Code duplication

```
1 def add():
2     p1 = convert_to_number(pop())
3     p2 = convert_to_number(pop())
4     ...
5
6 def sub():
7     p1 = convert_to_number(pop())
8     p2 = convert_to_number(pop())
9     ....
```

## SOLID (SRP, OCR, LSP, ISP, DIP), YAGNI

```
1 class Forth( object ):
2     def __init__( self , statements ):
3
4     def put( self ):
5
6     def pop( self ):
7
8     def add( self ):
9
10    def sub( self ):
11
12    def print_( self ):
13
14    def run( self ):
15
16 def execute( file_name ):
17     Forth( source_code ).run()
```

```
1 class Forth( __Stack__ ):
2     def put( self ):
3         pass
4
5     def pop( self ):
6         pass
```

Lexer => Parser => Compiler => Executor

Lexer + Parser

```
1  def parse_file(fd):
2      for lineno, raw_line in enumerate(fd):
3          line = raw_line.strip()
4          if line == "" or line.startswith("#"):
5              continue
6          try:
7              if " " not in line:
8                  yield lineno, (line, None)
9              else:
10                 cmd, param = line.split(" ", 1)
11                 if param.startswith('"') and param.endswith('
12                     param = param[1:-1]
13                 else:
14                     try:
15                         param = int(param)
16                     except ValueError:
17                         param = float(param)
```

```
18         yield lineno , (cmd, param)
19     except Exception as x:
20         print >>sys.stderr , "Parse error at line", lineno
21         raise
```

## Compiler + Executor

```
1 def execute(stack, fd):
2     for lineno, (cmd, param) in parse_file(fd):
3         if cmd == "put":
4             stack.append(param)
5         elif cmd == "add":
6             stack.append(stack.pop() + stack.pop())
7         elif cmd == "sub":
8             stack.append(stack.pop() - stack.pop())
9         elif cmd == "print":
10            print stack.pop()
11        else:
12            raise ValueError("Unknown command {} at line {}".format(cmd, lineno))
```

## Command

```
1 class Command(object):
2     name = None
3     param_count = None
4     minimum_stack_size = None
5     def __init__(self, **params):
6         self.params = params
7     def execute(self, stack):
8         if len(stack) < self.minimum_stack_size:
9
10 class Add(Command):
11     name = 'add'
12     param_count = 0
13     minimum_stack_size = 2
14     def execute(self, stack):
15         stack.append(stack.pop() + stack.pop())
16
17 all_commands = {Add.name: Add, ...}
```

## Command

```
1 class Command_0_0(object):
2     def __init__(self, **params):
3         self.name = params
4
5     def execute(self, stack):
6         if len(stack) < self.minimum_stack_size:
7             .....
8
9 class Add_0_2(Command):
10     pass
11
12 all_command_classes = [Add_0_2, ..]
13 all_commands = {}
```



## Command

```
1 for cmd in all_command_classes:
2     uname, stack_sz, num_params = cmd.__name__.split("_")
3     cmd.name = uname.lower()
4     cmd.param_count = int(num_params)
5     ...
6     all_commands[cmd.name] = cmd
```

## Command

```
1 def add_2_0( stack ):
2     # ....
3
4 def add_2_0( stack , s1 , s2 ):
5     # ...
6
7 def put( stack , p1 ):
8     stack.append(p1)
```