Code Smells

What is thta smell? Did you write that code?

Agenda

Day-1
Conventions 2hrs
Comprehension 30 minutes
Iterators and generators 1hr
decorators 1 hr
contextlib 30 minutes

Day-2
Descriptors 1 hrs
Unit Testing 2hrs
MetaClasses 1 hrs
Patterns 1 hrs

Day -3
Patterns 4hrs

Code Smell

CODE SMELLS ARE SYMPTOMS OF POOR DESIGN OR IMPLEMENTATION CHOISES

[Martin Fowler]

```
for i in range(0,len(InputTextboxes)):
            if HORI_ASSO_FLG==0:
                min_d =10000
                for j in range(0,len(TextLabelboxes)):
                    d= np.linalg.norm(InputTextboxes[i][0:2]-np.array(TextLabelboxes[j][0:2]))
                    if(d<min_d):
                        min_d =d
                        InputTextboxesMapping[i][0]=j
                        InputTextboxesMapping[i][1]=d
                if(min_d > 1.5* InputTextboxes[i][3]):
                    Label.append([0,0,0,0])
                else:
                    Label.append(TextLabelboxes[np.int(InputTextboxesMapping[i][0])])
```

Solution to the spaghetti code problem

Tip:
Any fool can write code that a computer can understand.
Good programmers write code that humans can understand.

- Refactoring

```
def getCordinates():
    if(count == 0):
        if(x_incr < 0 ):
            x1 = str(x - 50)
            x2 = str(x - 50)
        else:
            x1 = str(x + 150)
            x2 = str(x + 150)

        y1=str(y - 50)
        ty2=str(y)
    elif(x_incr > 0) :
        x1 = str(x)
        x2 = str(x+ x_incr)
        y1=str(y + 25)
        ty2=str(y + 25)
    else :
        x1 = str(x- x_incr)
        x2 = str(x)
        y1=str(y + 25)
        ty2=str(y + 25)
    return x1,y1,x2,ty2
```

Boy Scout Rule

Leave your code better than you found it.

Problem Accumulation

Area was pristine clean for many days, one fine day a lazy guy threw garbage at the place just one bag, next one week place was filled with garbage, reason other people saw garbage and instead of picking it up they threw their own.

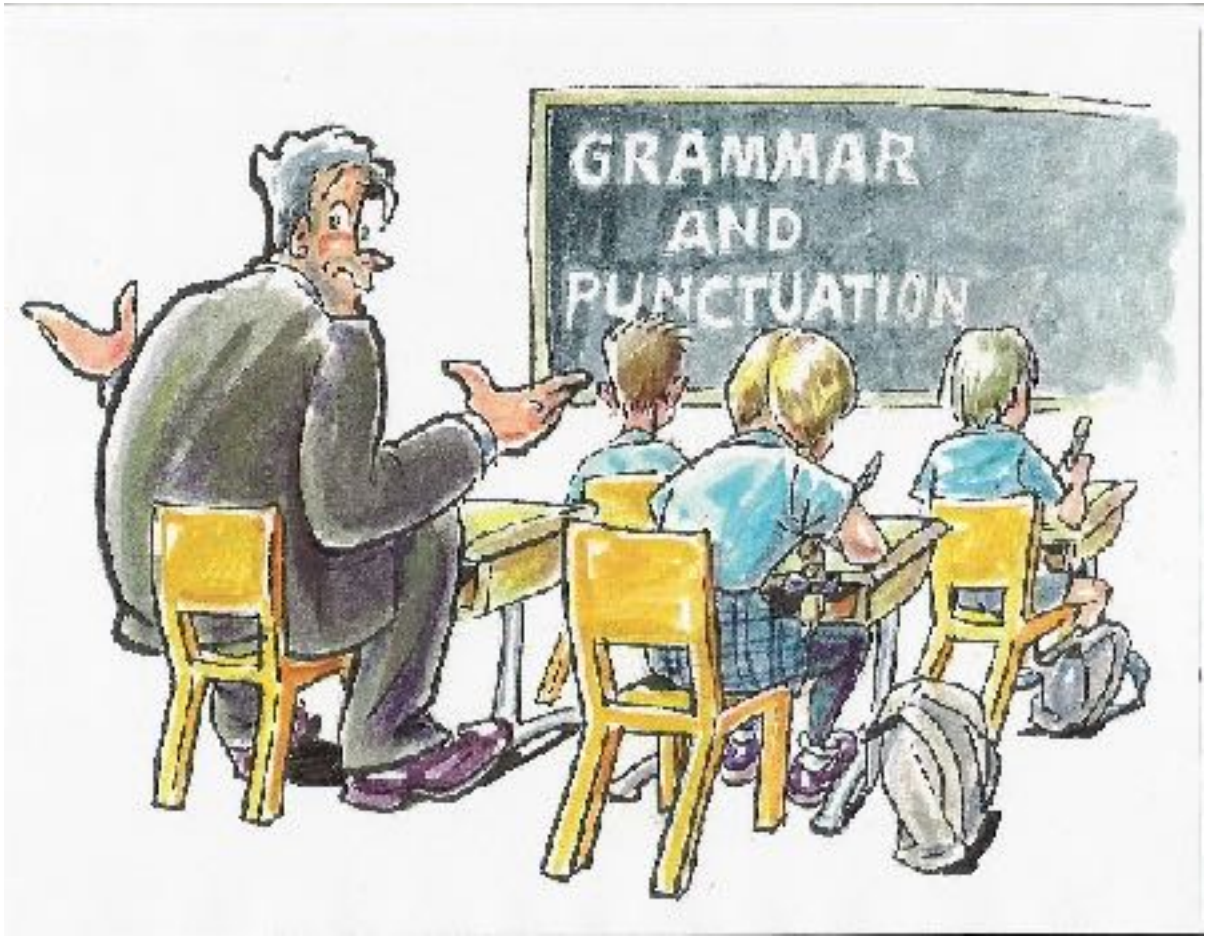
Comments

```
#Add Books associated with the Category
def add(category):
        bookMap.put(category.getCategoryId(), category.getBooks())
```

Explain Yourself in Code

```
def addBooksFromCategory(category):
        bookMap.put(category.getCategoryId(), category.getBooks())
```

```
patientCount+=1  # add patient hospital visiting
```



A comment worth writing is worth writing well.

```
i+=1 # increment I
```

Comments are Failures

they compensate for our inability to express in code

```
response = InputStreamResponse()
response.setBody(formatter.getResultStream(), formatter.getByteCount())
#resultsStream = formatter.getResultStream()
#reader = new StreamReader(resultsStream)
#response.setContent(reader.read(formatter.getByteCount()))
```

```
'''

def registerItem(item):
        prevElement= element
        count = count + 1
        if(count > 4):
            y = y + 100
            if(x_incr > 0) :
                x_incr = -100
            else:
                x_incr = 100
            count  = 0
        y2 = y + 50

}

….

'''
```

It is inappropriate for a comment to hold information better held in a as source code control system, issue tracking system, or any other record-keeping system.

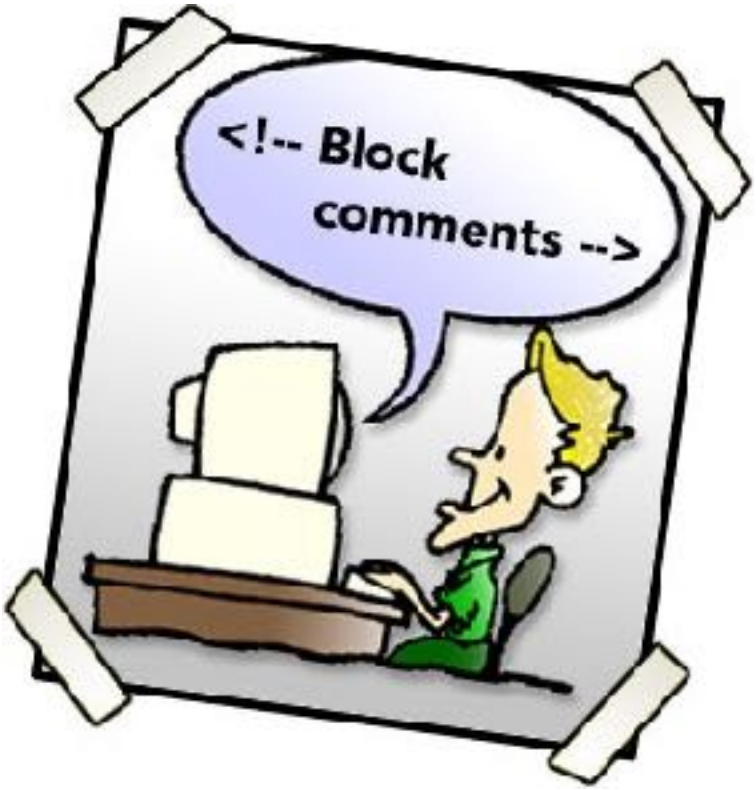
```
#
# @dateModified 12/12/1947
# @modifiedBy RamgopalVerma
# @modifiedReason because sky is so high
#
class Patient:
    pass
```

Avoid *Obsolete Comment*

It is best not to write a comment that will become obsolete.

They become floating islands of irrelevance and misdirection in the code.

```
#
# Returns the day of the month.
#
# @return the day of the month.
#
Def getDayOfMonth():
   return dayOfMonth
```



Comments should say things that the code cannot say for itself.

```
# Check to see if the employee is eligible for full benefits
if (employee.flags and HOURLY_FLAG) and (employee.age > 65):
```

Explain Yourself in Code

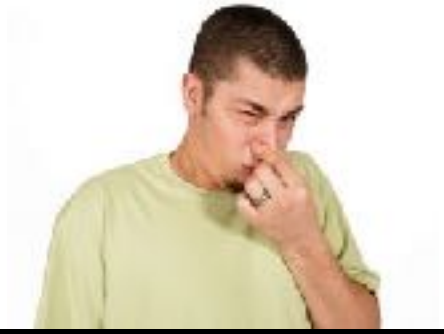
```
if employee.isEligibleForFullBenefits():
```

```
def getExpenseLimit() :
  # should have either expense limit or a primary project
  return _expenseLimit if_expenseLimit != NULL_EXPENSE
                       else  _primaryProject.getMemberExpenseLimit()
```

Introduce Assertion

```
def getExpenseLimit():
  assert _expenseLimit != NULL_EXPENSE or _primaryProject != null
  return _expenseLimit if_expenseLimit != NULL_EXPENSE
                       else  _primaryProject.getMemberExpenseLimit()
```

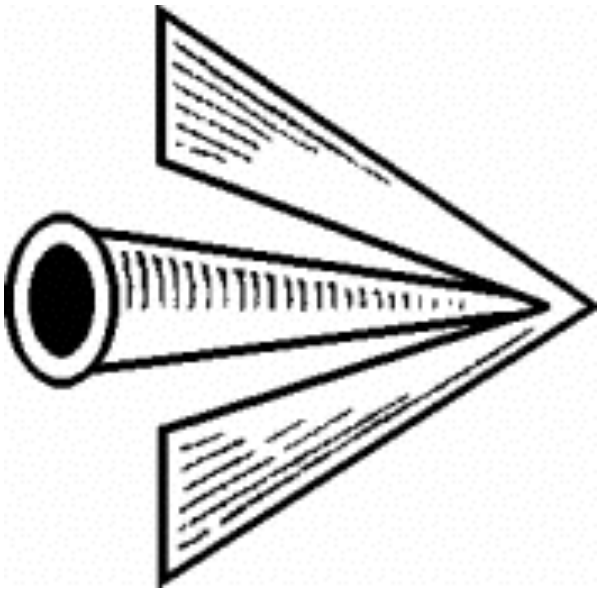
Error



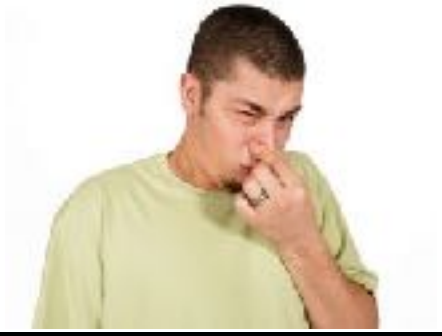
```
def DoJob():
        domain = new Domain()
        res = domain.fun()
        if res == true:
                res = domain.fun2(100)
                if res == true:
                        rep = new Repository()
                        emp = rep.get(1)
                        if emp != ' ':
                                ...
                        else:
                                …

                else:
                        …

        else:
                …
```

```
public void sendShutDown() {
      DeviceHandle handle = getHandle(DEV1);
      // Check the state of the device
      if (handle != DeviceHandle.INVALID) {
            // Save the device status to the record field
            retrieveDeviceRecord(handle);
            // If not suspended, shut down
            if (record.getStatus() != DEVICE_SUSPENDED) {
                  pauseDevice(handle);
                  clearDeviceWorkQueue(handle);
                  closeDevice(handle);
            } else {
                  logger.log("Device suspended. Unable to shut down");
            }
      } else {
               logger.log("Invalid handle for: " + DEV1.toString());
      }
}
```

Arrow Code



```
if
  if
    if
      if
        do something
      endif
    endif
  endif
endif
```



```
def DoJob():
        domain = new Domain();
        domain.fun();
        domain.fun2(100);
        rep = new Repository();
        emp = rep.get(1);
        ...


        try:
                DoJob();
        except Exception as e:
                ….
```

Use Exceptions Rather Than Return Codes

```
public class DeviceController {
      ...
      public void sendShutDown() {
            try {
                  tryToShutDown();
            } catch (DeviceShutDownError e) {
                  logger.log(e);
            }
      }
      private void tryToShutDown() throws DeviceShutDownError {
            DeviceHandle handle = getHandle(DEV1);
            DeviceRecord record = retrieveDeviceRecord(handle);
            pauseDevice(handle);
            clearDeviceWorkQueue(handle);
            closeDevice(handle);
      }
      private DeviceHandle getHandle(DeviceID id) {
            ...
            throw new DeviceShutDownError("Invalid handle for: " + id.toString());
            ...
      }
      ...
}
```

```
def withdraw(amount) :
  if amount > _balance:
    return -1
  else:
    _balance -= amount
    return 0
```

Replace Error Code with Exception

```
def withdraw(amount) :
  if amount > _balance:
        raise Exception(“no sufficient balance”)

  _balance -= amount
```

Implement user defined exceptions

```
class DivisorTooSmallError(StandardError):
    def __init__(self, arg):
        self.args = arg


def divide(a, b):
    if b < 1:
        raise DivisorTooSmallError
    return a / b


try:
    divide(10, 0)
except DivisorTooSmallError:
    print("Unable to divide these numbers!")
```

```
def getValueForPeriod (periodNumber):
        if periodNumber >= len(_values):
                return 0

        return _values[periodNumber]
```

easier to ask for forgiveness than permission (EAFP)

```
def getValueForPeriod (periodNumber):
  try:
        return _values[periodNumber]
  except ArrayIndexOutOfBoundsException:
        return 0
```

Look Before You Leap (LBYL) v/s *Easier to Ask for Forgiveness than Permission* (EAFP)

- if hasattr(someobj, 'open'): [...] else: [...]

- try: someobj.open() [...] except AttributeError: [...]

Duck typing is a central idea in Python

```
def print_object(some_object):
    # Check if the object is printable...
    try:
        printable = str(some_object)
        print(printable)
    except TypeError:
        print("unprintable object")
```

```
try:
   while AccountManager.HasMoreAccounts():

        account = AccountManager.GetNextAccount()
        if account.Name == userName:
           #We have it
           raise Exception(“Account Found”)

except Exception:
   print("Here are your account details: " + found.Account.Details.ToString())
```

Replace Exception with Test

```
Account found = None;
while AccountManager.HasMoreAccounts() and (found is None):
    account = AccountManager.GetNextAccount()
    if account.Name == userName:
        #We found it
        found = account

If found:
        print("Here are your account details: " + found)
```

```
try :
    f = File.Open(myfile);
    #do something

except Exception e:
    print("Could not open file”)
```

Always catch specific Exception

```
try :
    f = File.Open(myfile);
    #do something

except IOException e:
    print("Could not open file”)
except  UnauthorizedAccessException e:
     print("Insufficient rights")
except Exception as e:
     # handle any other exception
     print("Error '{0}' occured. Arguments {1}.".format(e.message, e.args))
else:
    # Excecutes if no exception occured
    print("No errors")
finally:
   # Executes always
```

```
f = open('afile.txt', 'w')
try:
    n = f.write('sometext')
finally:
    f.close()
```

```
with open('afile.txt', 'w') as f:
    n = f.write('sometext')
```

```
class MongoDBConnectionManager():
    def __init__(self, hostname, port):
        self.hostname = hostname
        self.port = port
        self.connection = None

    def __enter__(self):
        self.connection = MongoClient(self.hostname, self.port)
        return self

    def __exit__(self, exc_type, exc_value, exc_traceback):
        self.connection.close()


with MongoDBConnectionManager('localhost', '27017') as mongo:
    collection = mongo.connection.SampleDb.test
    data = collection.find({'_id': 1})
    print(data.get('name'))
```

```
import contextlib

class Door(object):
    def __init__(self):
        print ('  __init__()')
    def close(self):
        print ('  close()')

with contextlib.closing(Door()) as door:
    print ('  inside with statement')
    raise RuntimeError('error message')
```

```
from contextlib import contextmanager

@contextmanager
def open_file(path, mode):
    the_file = open(path, mode)
    yield the_file
    the_file.close()

files = []

for x in range(100000):
    with open_file('foo.txt', 'w') as infile:
        files.append(infile)
```

```
try:
    5 / 0
except Exception as e:
    print("Exception")
# unreachable code!
except ZeroDivisionError as e:
    print("ZeroDivisionError")
```

Move sub class exception clause before its ancestor's clause

```
try:
    5 / 0
except ZeroDivisionError as e:
    print("ZeroDivisionError")
except Exception as e:
    print("Exception")
```

```
def readLines(fileName) :
    try :
      f = open('afile.txt', ‘w')
      …
    excpet Exception e:
      return 0

    return list
```

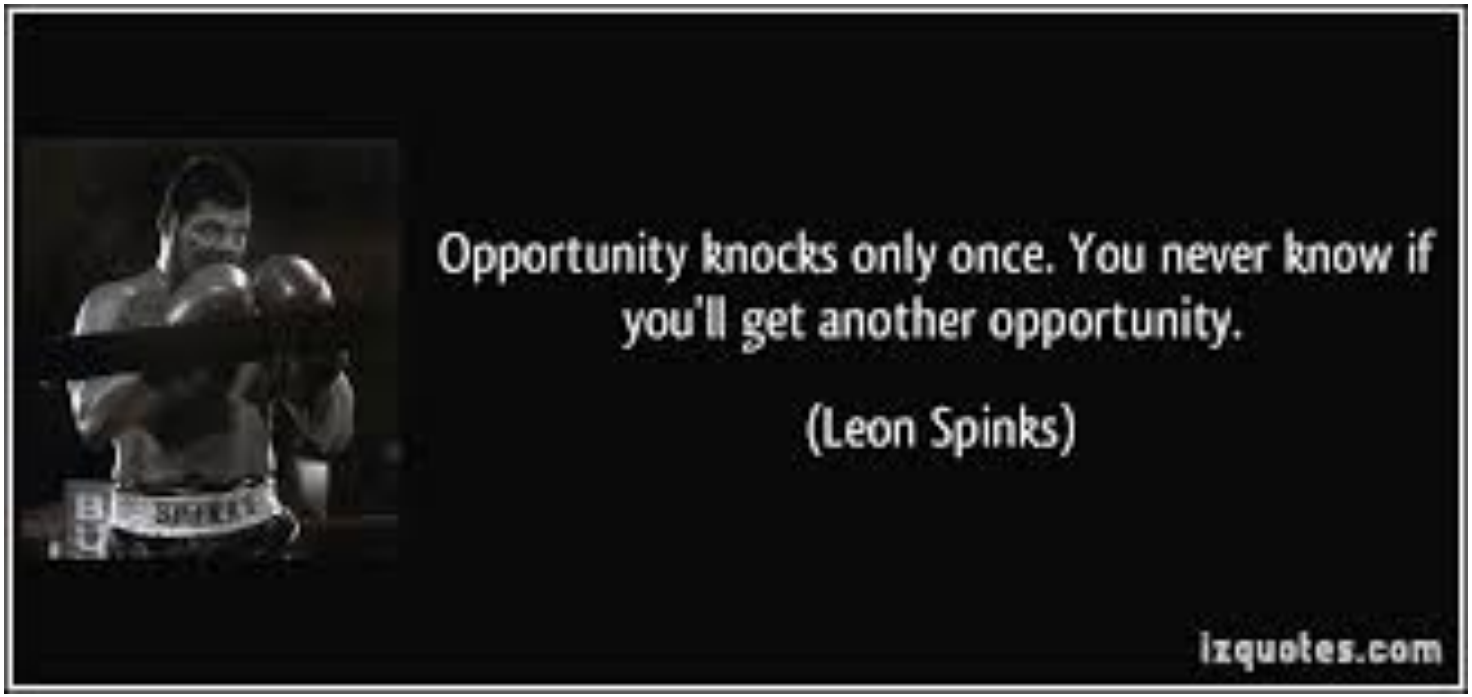
Always know why you are catching an exception

If you don't catch the exception, you get a traceback.

```
try:
    #logic

except Exception e:
    Pass
```

```
#method 1
except SQLException exp:
  log.fatal("Exception occurred:", exp)
  raise exp

#method 2
try:
  #call method 1
except Exception exp:
  log.fatal("Exception occurred:", exp)
```

Log exception stack trace only once



```
catch(InterruptedException e) {
   AppException exp = new AppException();
   exp.addError(Constants.ERROR_CODE, "EXP005");
   exp.addError(Constants.ERROR_MESG, e.getMessage());
   throw exp;
}
catch(SQLException e) {
   AppException exp = new AppException();
   exp.addError(Constants.ERROR_CODE, "EXP011");
   exp.addError(Constants.ERROR_MESG, e.getMessage());
   throw exp;
}
catch(Exception e) {
   AppException exp = new AppException();
   exp.addError(Constants.ERROR_CODE, "EXP003");
   exp.addError(Constants.ERROR_MESG, e.getMessage());
   throw exp;
}
```

remove duplicate code

```
private AppException createAppException(String errorCode,e) {
   AppException exp = new AppException();
   exp.addError(Constants.ERROR_CODE, errorCode);
   exp.addError(Constants.ERROR_MESG, e.getMessage());
   return exp;
}

//code
catch(InterruptedException exp) {
   throw createAppException("EXP005",exp);
}
catch(SQLException exp) {
   throw createAppException("EXP011",exp);
}
catch(Exception exp) {
   throw createAppException("EXP003",exp);
}
```

```
logger.error("Exception occurred:" + e.message)
```

```
import logging
logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger(__name__)

try:
    do_something_that_might_error()
except Exception as error:
    logger.exception(error)
```

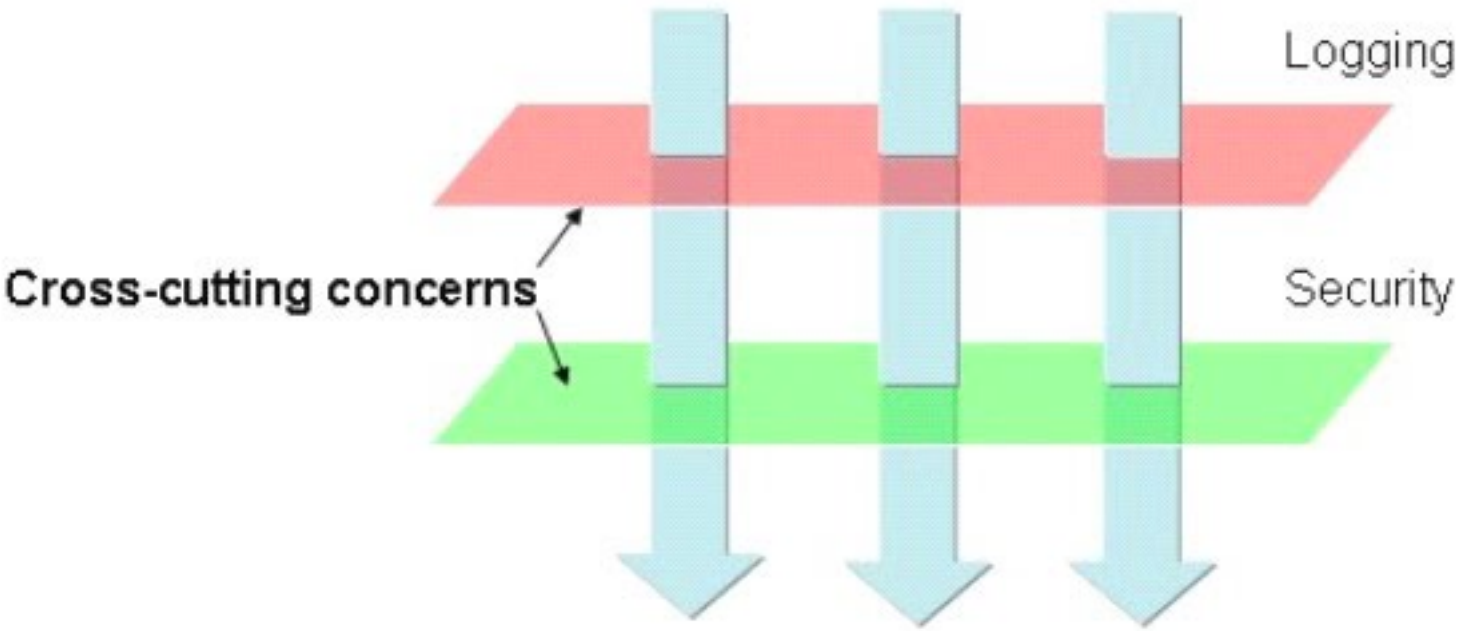
the exception stacktrace is lost

```
def fun():
    try:
        …logic 1

        if cond:
            raise Exception(…);

        …logic 2

    except Exception e:

        …logic 3
```

Farmers in India use Pepsi as pesticides, because it's cheaper than pesticides and gets the job done just as well.

```
try {
      …

} catch (Exception e) {
        LOGGER.error("Exception :", e);
        LOGGER.debug("Exception :" + e.getMessage());
        try {
                con.rollback();
        } catch (SQLException e1) {
                LOGGER.error("Exception :", e1);
                LOGGER.debug("Exception :" + e1.getMessage());
        }
} finally {
        try {
                con.close();
        } catch (SQLException e1) {
                LOGGER.error("Exception :", e1);
                LOGGER.debug("Exception :" + e1.getMessage());
        }
}
```

```
def api1() :
    try :
      #api 1 logic
    excpet Exception e:
      #error logic

def api2() :
    try :
      #api 2 logic
    excpet Exception e:
      #error logic

def api3() :
    try :
      #api 3 logic
    excpet Exception e:
      #error logic
```


Logging
Cross-cutting concerns
Security

```
def log(func):
    def wrapper():
        print("before")
        func()
        print("after")
    return wrapper

def dojob():
    print ("I'm a normal function.")

f = log(dojob)
f()
```

```
def log(func):
    def wrapper():
        print("before")
        func()
        print("after")
    return wrapper

@log
def dojob():
    print ("I'm a normal function.")

dojob()
```

```
def log(func):
    def wrapper(x,y):
        print("before")
        res = func(x,y)
        print("after")
        return res
    return wrapper

def add(x,y):
    print ("add")
    return x + y

f = log(add)
res = f(10,20)
print(res)
```

```
def func1(x, y, z):
    print (x)
    print (y)
    print (z)

def func2(*args):  ←   packing
    args[0] = 'Hello'
    args[1] = 'awesome'
    func1(*args)  ←   unpacking

func2('Goodbye', 'cruel', 'world!')
```

```
def fun(*args):
    arg1, arg2 = args
    print ("arg1: %r, arg2: %r" % (arg1, arg2))



fun("Zed","Shaw")

lst = ["Zed","Shaw"]
fun(*lst)
```

```
def fun(**kargs):
    print(kargs)

fun(x="Zed",y="Shaw")

dct = {'x': "Zed", 'y':"Shaw"}

fun(**dct)
```

```
def log(func):

    def wrapper(*args, **kwargs):
        print ("Arguments were: %s, %s" % (args, kwargs))
        res = func(*args, **kwargs)
        print("after")
        return res
    return wrapper

def add(x,y):
    print ("add")
    return x + y

f = log(add)
res = f(10,20)
print(res)
```

```
def log(data):
    def log2(func):
        def wrapper(*args, **kwargs):
            print ("Arguments were: %s, %s" % (args, kwargs))
            res = func(*args, **kwargs)
            print("after")
            return res
        return wrapper
    return log2

@log("hello")
def add(x,y):
    print ("add")
    return x + y

res = add(10,20)
print(res)
```

```
def entryExit(f):
 def new_f():
        print ("Entering", f.__name__)
        f()
        print ("Exited", f.__name__)
    return new_f

@entryExit
def func1():
    print ("inside func1()")

@entryExit
def func2():
    print ("inside func2()")
```

functions

Dead Function

Methods that are never called.

Ruthlessly delete code that isn't being used.

```
double potentialEnergy(double mass, double height) {
  return mass * height * 9.81;
}
```

Magic Numbers and Strings

Explicit is better than implicit. - Tim Peters, The Zen of Python

Replace Magic Number with Symbolic Constant

```
double potentialEnergy(double mass, double height) {
  return mass * GRAVITATIONAL_CONSTANT * height;
}
static final double GRAVITATIONAL_CONSTANT = 9.81;
```

```
class Constants
{
    // Prevents instanciation of myself and my subclasses
    private Constants() {}

    public final static String TOTO = "toto";
    public final static Integer TEN = 10;
    public final static String NAME=“name1”;
    public final  static int MAX_VAL=25;
    //...
}
```

- DO NOT create a dreadful 4000-lines Constants class !

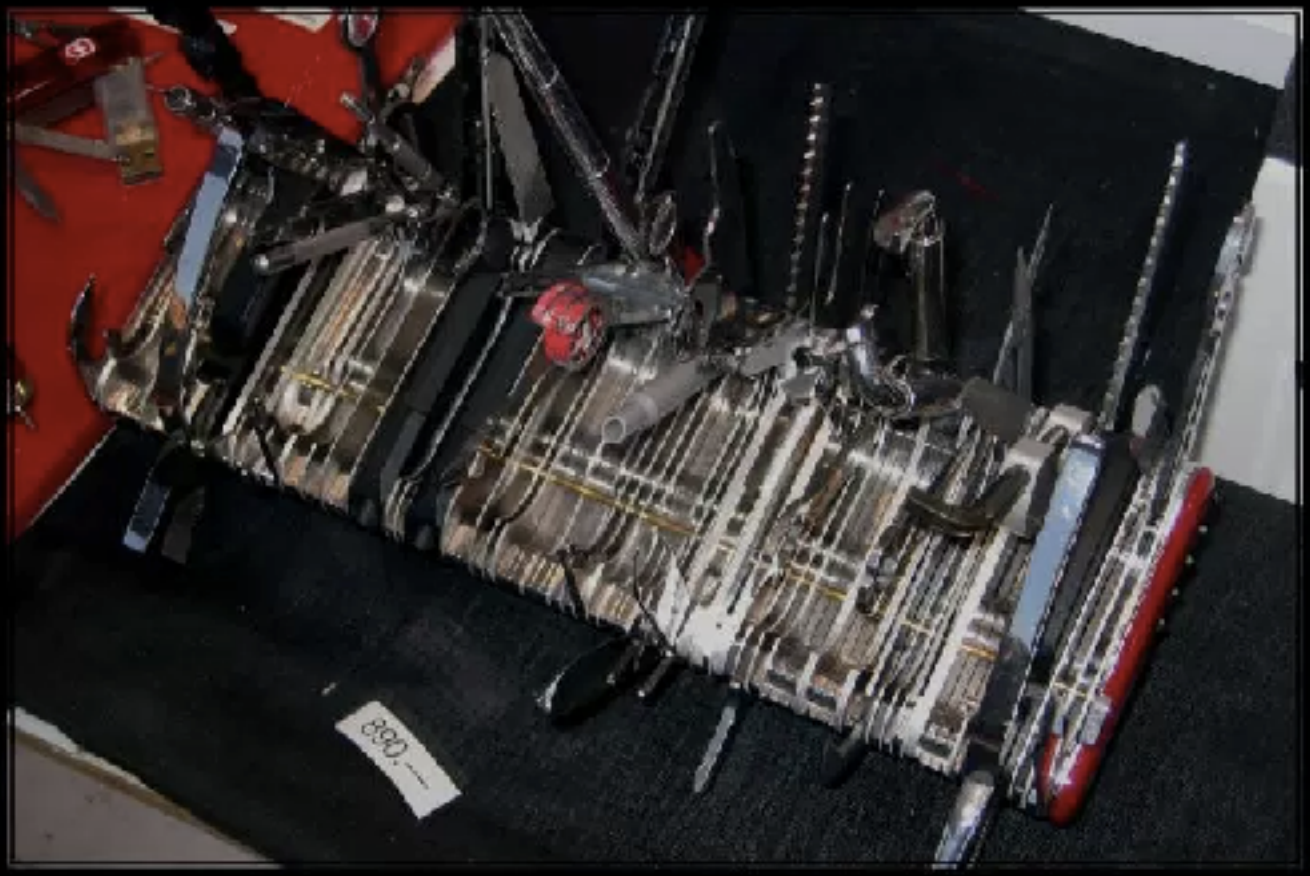
```
while ((!found) && (pos < (fileContent.Length - 6)))
{
    byteData = new byte[6];
    Array.Copy(fileContent, pos, byteData, 0, 6);
    pos = pos + 6;
    str_byteData = enc.GetString(byteData);
    if (str_byteData.Contains("s"))
    {
        posE_byteData = str_byteData.IndexOf("s");
        pos = pos + (posE_byteData - 6);
        Array.Copy(fileContent, pos, byteData, 0, 6);
        pos = pos + 6;
        if (byteData[0] == 0x73)    // 's'
        {
            if (byteData[1] == 0x74)    // 't'
            {
                if (byteData[2] == 0x72)    // 'r'
                {
                    if (byteData[3] == 0x65)    // 'e'
                    {
                        if (byteData[4] == 0x61)    // 'a'
                        {
                            if (byteData[5] == 0x6D)    //
                            {
                                found = true;
                                break;
                            }
                            else
                            {
                                if (byteData[5] == 0x73)
                                {
                                    pos = pos - 1;
```

Nesting Levels

{if < 4 , switch,while,do, for <3}

```
        FormView1.FindControl("noteRow").Visible = true;
        FormView1.FindControl("RequiredFieldValidator4").Visible = false;
        // TODO: Fix this - What if Action is 2nd parameter in URL
        if (Request.Path.Contains(@"clientProfile.aspx?Action=Add"))
        {
            FormView1.FindControl("thLoginID").Visible = true;
            FormView1.FindControl("tdLoginID").Visible = true;
            FormView1.FindControl("txtLoginID").Visible = true;
        }

    }
    else
    {
        FormView1.FindControl("noteRow").Visible = false;
    }


if (HttpContext.Current.User.IsInRole("Administrator"))
{

    if (!Request.Path.Contains(@"/administratorProfile.aspx"))
    {
        if (!Request.Path.Contains(@"/Profile.aspx"))
        {
            FormView1.FindControl("txtLoginID").Visible = false;
            FormView1.FindControl("lblLoginID").Visible = false;
            -- Snip snip snip - more controls are hidden or shown -- --

            if (Request.Path.Contains(@"/UserProfile.aspx") && HttpContext.Current.User.IsInRole("User") && (Request.QueryString["Action"] == null ||
            {

                FormView1.FindControl("thLoginID").Visible = true;
                FormView1.FindControl("lblLoginID").Visible = true;
                -- Snip snip snip - more controls are hidden or shown -- --

            }
            else
            {
                FormView1.FindControl("thLoginID").Visible = false;
                FormView1.FindControl("lblLoginID").Visible = false;
                -- Snip snip snip - more controls are hidden or shown -- --


            }

        }
        else
        {
            FormView1.FindControl("thDateRegistered").Visible = false;
            FormView1.FindControl("tdDateRegistered").Visible = false;
            -- Snip snip snip - more controls are hidden or shown -- --

        }
    }
    else if (Request.Path.Contains(@"/ReaderProfile.aspx"))
    {
        FormView1.FindControl("tdAdviser").Visible = false;
        FormView1.FindControl("thAdviser").Visible = false;
        -- Snip snip snip - more controls are hidden or shown -- --

    }
}
else if (HttpContext.Current.User.IsInRole("Administrator"))
```

Long Method



Single Responsibility Principle

Yes, it's all there. No, it's not a good idea.

Functions Should be Small !!

- How small ?
  - **4 *lines*** is best; 5 is ok; 6 is already too much
  - By any means, keep your functions smaller than a page of your IDE!

- Why so small ?!

➔To be sure that they do only 1 THING

  - ➔In a few lines, you can't do much
  - ➔Its name can then tell you everything

```
class Boiler:
    def safety_check(self):
        # Convert fixed-point floating point
        temperature = self.modbus.read_holding()
        pressure_psi = self.abb_f100.register / F100_FACTOR
        if psi_to_pascal(pressure_psi) > MAX_PRESSURE:
            if temperature > MAX_TEMPERATURE:
                # Shutdown!
                self.pnoz.relay[15] &= MASK_POWER_COIL
                self.pnoz.port.write('$PL,15\0')
                sleep(RELAY_RESPONSE_DELAY)
                # Successful shutdown?
                if self.pnoz.relay[16] &amp;amp; MASK_POWER_OFF:
                    # Play alarm
                    with open(BUZZER_MP3_FILE) as f:
                        play_sound(f.read())
```

Divide program into methods that perform one task

```
# Better
class Boiler:
    def safety_check(self):
        if any([self.temperature > MAX_TEMPERATURE,
                self.pressure > MAX_PRESSURE]):
            if not self.shutdown():
                self.alarm()

    def alarm(self):
        with open(BUZZER_MP3_FILE) as f:
            play_sound(f.read())

    @property
    def pressure(self):
        pressure_psi = abb_f100.register / F100_FACTOR
        return psi_to_pascal(pressure_psi)
    ...
```

```
public class UglyMoneyTransferService
{
public void transferFunds(Account source, Account target, BigDecimal amount, boolean allowDuplicateTxn){
        Account sourceAccount = null;
        if(rs.next()) {
            sourceAccount = new Account();
            //populate account
        }
        Account targetAccount = null;
        if(!sourceAccount.isOverdraftAllowed()) {
            if((sourceAccount.getBalance() - amount) < 0) {
                throw new RuntimeException("Insufficient Balance");
            }
        }
        else {
            if(((sourceAccount.getBalance()+sourceAccount.getOverdraftLimit()) - amount) < 0) {
                throw new RuntimeException("Insufficient Balance, Exceeding Overdraft Limit");
            }
        }
        AccountTransaction lastTxn = .. ; //JDBC code to obtain last transaction of sourceAccount
        if(lastTxn != null) {
            if(lastTxn.getTargetAcno().equals(targetAccount.getAcno()) && lastTxn.getAmount() == amount && !allowDuplicateTxn) {
                throw new RuntimeException("Duplicate transaction exception");
            }
        }
            sourceAccount.debit(amount);
            targetAccount.credit(amount);
        }

}
```

```
Public void doTheDomesticThings()
{
        takeOutTheTrash();
        walkTheDog();
        for(Dish dish : dirtyDishStack)
        {
                sink.washDish(dish);
                teaTowel.dryDish(dish);
        }
}
```

Don’t mix levels of abstraction

```
Public void doTheDomesticThings()
{
        takeOutTheTrash();
        walkTheDog();
        doTheDishes();
}
```

```
public List<ResultDto> buildResult(Set<ResultEntity> resultSet) {
    List<ResultDto> result = new ArrayList<>();
    for (ResultEntity entity : resultSet) {
        ResultDto dto = new ResultDto();
        dto.setShoeSize(entity.getShoeSize());
        dto.setNumberOfEarthWorms(entity.getNumberOfEarthWorms());
        dto.setAge(computeAge(entity.getBirthday()));
        result.add(dto);
    }
    return result;
}
```

Don’t mix levels of abstraction

```
public List<ResultDto> buildResult(Set<ResultEntity> resultSet) {
    List<ResultDto> result = new ArrayList<>();
    for (ResultEntity entity : resultSet) {
        result.add(toDto(entity));
    }
    return result;
}

private ResultDto toDto(ResultEntity entity) {
    ResultDto dto = new ResultDto();
    dto.setShoeSize(entity.getShoeSize());
    dto.setNumberOfEarthWorms(entity.getNumberOfEarthWorms());
    dto.setAge(computeAge(entity.getBirthday()));
    return dto;
}
```

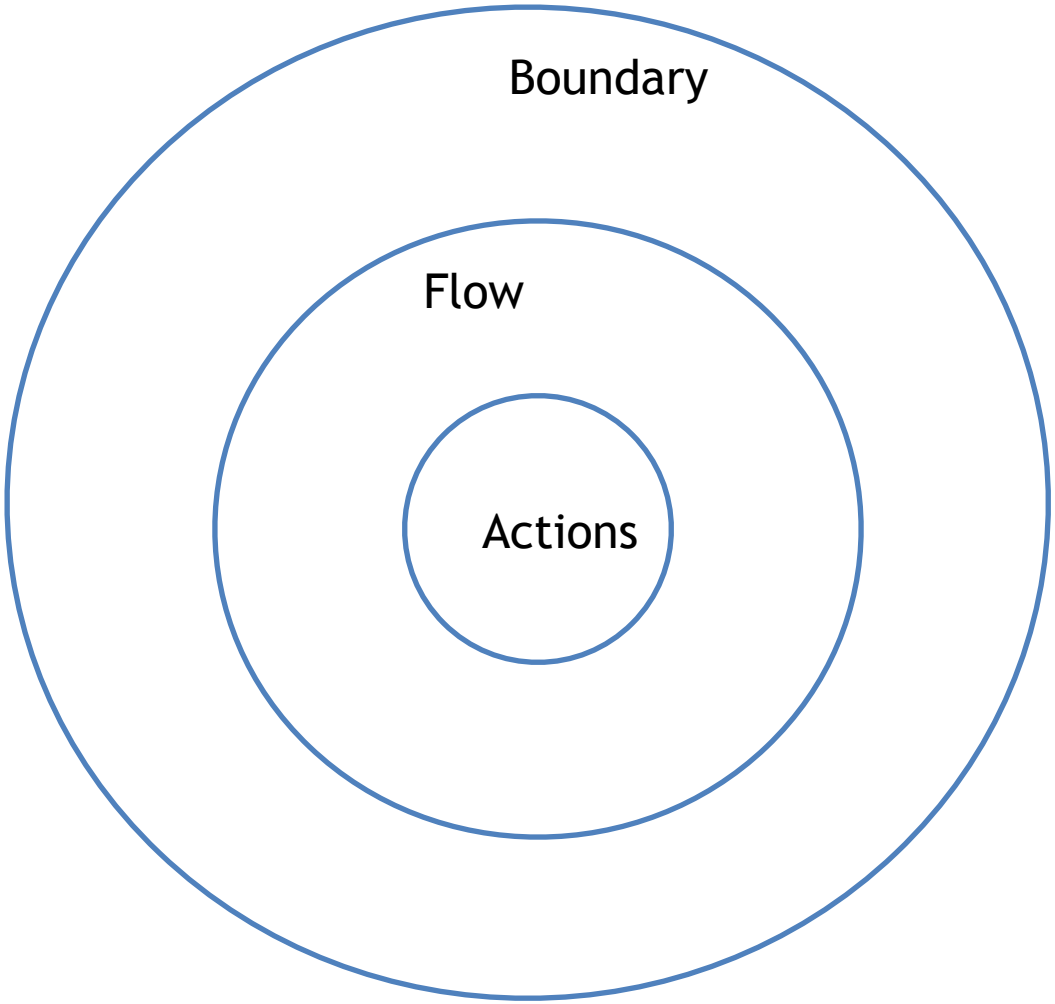
```
private Account validateAndGetAccount(String acno){
    Account account = AccountDAO.getAccount(acno);
    if(account == null){
    throw new InvalidAccountException("Invalid ACNO :"+acno);
    }
}
private void checkForOverdraft(Account account, BigDecimal amount){
    if(!account.isOverdraftAllowed()){
        if((account.getBalance() - amount) < 0){
            throw new InsufficientBalanceException("Insufficient Balance");
        }
    }
    else{
        if(((account.getBalance()+account.getOverdraftLimit()) - amount) < 0){
            throw new ExceedingOverdraftLimitException("Insufficient Balance, Exceeding Overdraft Limit");
        }
    }
}
private void checkForDuplicateTransaction(FundTransferTxn txn){
    AccountTransaction lastTxn = TransactionDAO.getLastTransaction(txn.getSourceAccount().getAcno());
        if(lastTxn != null){
            if(lastTxn.getTargetAcno().equals(txn.getTargetAccount().getAcno())
            && lastTxn.getAmount() == txn.getAmount()
            && !txn.isAllowDuplicateTxn()){
                throw new DuplicateTransactionException("Duplicate transaction exception");
        }
    }
}
private void makeTransfer(Account source, Account target, BigDecimal amount){
    sourceAccount.debit(amount);
    targetAccount.credit(amount);
    TransactionService.saveTransaction(source, target,  amount);
}
```

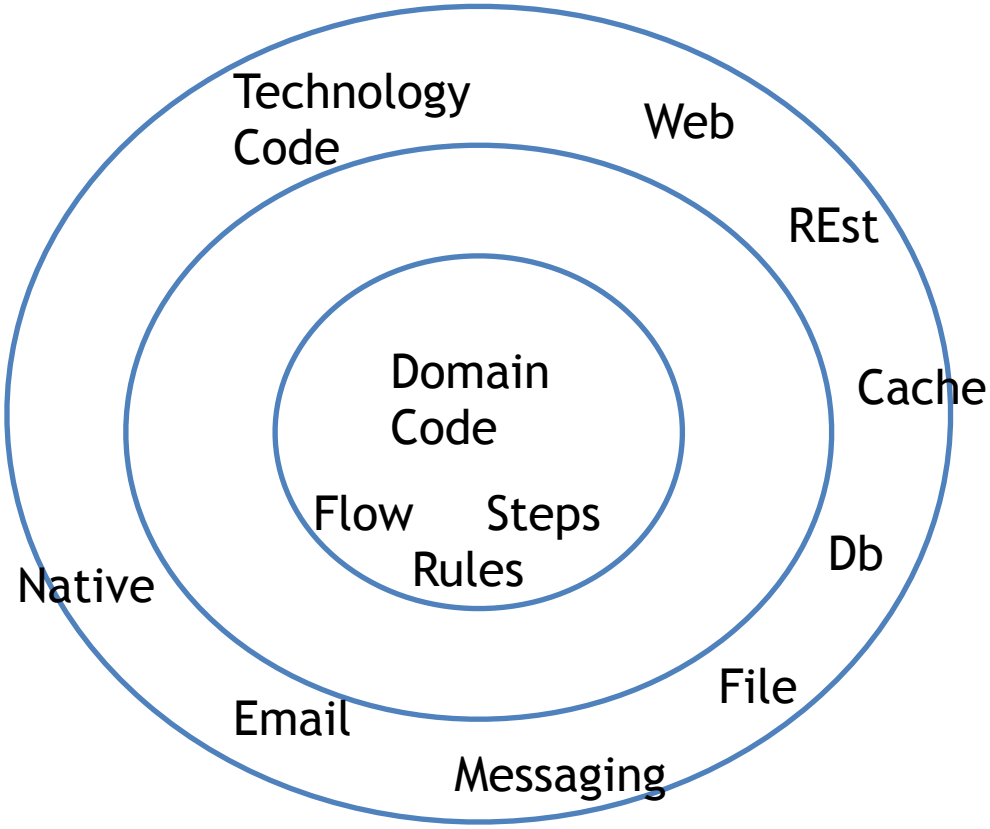
```
class FundTransferTxn
{
private Account sourceAccount;
private Account targetAccount;
private BigDecimal amount;
private boolean allowDuplicateTxn;
//setters & getters
}



public class CleanMoneyTransferService
{
    public void transferFunds(FundTransferTxn txn) {
        Account sourceAccount = validateAndGetAccount(txn.getSourceAccount().getAcno());
        Account targetAccount = validateAndGetAccount(txn.getTargetAccount().getAcno());
        checkForOverdraft(sourceAccount, txn.getAmount());
        checkForDuplicateTransaction(txn);
        makeTransfer(sourceAccount, targetAccount, txn.getAmount());
    }

    ....
}
```

```
public WebUser getCurrentUser(){
        if (_currentUser == null) {
            Object obj  = HttpContext.Current.Session["__currentUser"];
            if (obj != null) {
                _currentUser = (WebUser)obj;
                return _currentUser;
            }
            SecurityHelper secHelper = new SecurityHelper();
            WebUserRepository rep = new WebUserRepository();
            if (secHelper.TrackingGuid != Guid.Empty){
                _currentUser = rep.GetWebUserByTrackingGuid(secHelper.TrackingGuid);
                if (_currentUser != null)
               return _currentUser;
            }
            WebUserFactory factory = new WebUserFactory();
            _currentUser = factory.CreateWebUser();
        }
        return _currentUser;
}
```

Avoid multiple returns

```
private WebUser _currentUser;
 public WebUser getCurrentUser()
{
  if (_currentUser == null) _currentUser = GetWebUserFromSession();
     if (_currentUser == null) _currentUser = GetWebUserFromTrackingCookie();
     if (_currentUser == null) _currentUser = CreateNewWebUser();
     return _currentUser;
}
private WebUser GetWebUserFromSession()
{
     Object obj = HttpContext.Current.Session["__currentUser"];
     return obj == null ? null : (WebUser)obj;
}
private WebUser GetWebUserFromTrackingCookie()
{
     SecurityHelper secHelper = new SecurityHelper();
     WebUserRepository rep = new WebUserRepository();
     if (secHelper.TrackingGuid == Guid.Empty)
          return null;
     else
          return rep.GetWebUserByTrackingGuid(secHelper.TrackingGuid);
}
private WebUser CreateNewWebUser()
{
     WebUserFactory factory = new WebUserFactory();
     return factory.CreateWebUser();
}
```


Boundary
Flow
Actions


Technology
Code
Web
REst
Cache
Domain
Code
Flow
Steps
Rules
Db
Native
File
Email
Messaging

```
double getPrice()
{
        int basePrice = _quantity * _itemPrice;

        double discountFactor;
        if(basePrice > 1000)
                discountFactor = 0.95;
        else
                discountFactor = 0.98;

        return basePrice * discountFactor;
}
```

Replace Temp with Query

```
double getPrice()
{
        return basePrice() * discountFactor();
}
private double discountFactor()
{
        if(basePrice > 1000)
                return 0.95;
        else
                return  0.98;
}
private int basePrice()
{
        return _quantity * _itemPrice;
}
```

```
def CreateFile(name, isTemp):
   #implementation


myfile = CreateFile("foo.txt", true)
```

```
Def PrintReceipt(self,printDetails):
  for item in self.order:
    PrintLine(item.Name)
    if printDetails:
      PrintLine(item.Description)
```

SINGLE RESPONSIBILITY PRINCIPLE

Every object should have a single responsibility, and all its services should be narrowly aligned with that responsibility.

Flag Arguments are Ugly

```
customer.setLoggedIn(true);
```

```
removeOrders(customer, false, true);
```

- It shouts that the function does more than one thing !

Reveal Intend

```
def CreatePermenantFile(name):
    pass

def CreateTempFile(String name):
    pass
```

Reveal Intend

```
isTemp = True
CreateFile("foo.bar", isTemp)
```

If you have to call an existing API that uses this style,
you should introduce a local variable for clarity.

```
bool getPositiveNumbers(l1, l2) {

    for (Integer el1: l1)
        if (el1 > 0)
            l2.add(el1);

    return true;
}
```

Avoid output parameters

```
List<Integer> getPositiveNumbers(List<Integer> numbers) {
    return numbers.stream()
        .filter(num -> num > 0)
        .collect(toList());
}
```

- The caller may be surprised that an argument changed.

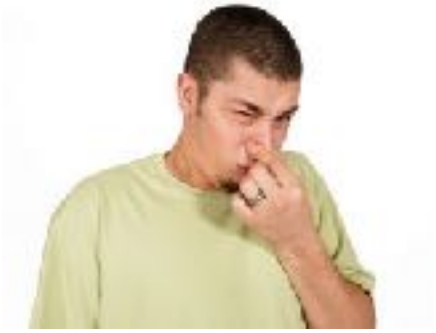
Did you expect that the second parameter will be changed when you call that method?

principle of least astonishment

the name of a function should reflect what it does

result of performing some operation should be obvious, consistent, and predictable, based upon the name of the operation

Embrace Immutability



```
myBadMethod("John","Michael",
      "Paris","St. Albergue");
```

Max 2 parameters

```
params.setStreet(“Paris");
…
myBadMethod(params);
```

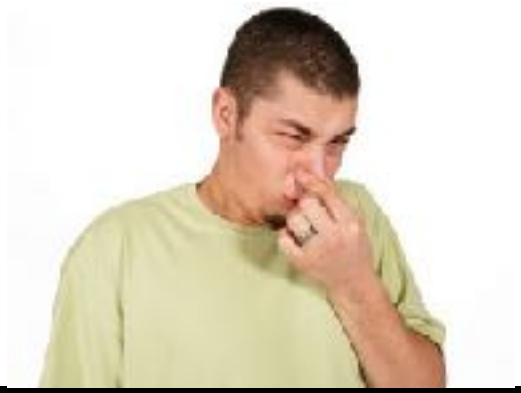
- For ≥ 3 parameters, it's hard to read: their order becomes a problem

➔ group them in a **Parameter Object**/**DTO**

```
void bad_handle_data(char *data, size_t length)
{
        if (check_CRC(data, length) == OK)
        {
                /* * 30 * lines * of * data * handling */
        }
        else
        {
                printf("Error: CRC check failed\n");
        }
}
```

Test Exception instead of normal code

```
void good_handle_data(char *data, size_t length)
{
        if (check_CRC(data, length) != OK)
        {
                printf("Error: CRC check failed\n");
                return;
        }

        /* * 30 * lines * of * data * handling */

}
```



```
List<Employee> employees = getEmployees();
if (employees != null) {
    for(Employee e : employees) {
        totalPay += e.getPay();
    }
}

….
```

```
Man findABrotherInLawOf(Man user) {
    Man anyBrotherInLaw = null;
    Woman userWife = user.getWife();

    if (userWife != null) {
        List<Man> brothers = userWife.getBrothers();

        if (brothers != null) {
            Battery firstBrother = brothers.get(0);

            if (firstBrother != null) {
                anyBrotherInLaw = firstBrother;
            }
        }
    }
   return anyBrotherInLaw;
}
```

```
public void registerItem(Item item) {
      if (item != null) {
            ItemRegistry registry = peristentStore.getItemRegistry();
            if (registry != null && registry.Count > 0) {
                  Item existing = registry.getItem(item.getID());
                  if (existing !=null && existing.getBillingPeriod().hasRetailOwner()) {
                        existing.register(item);
                  }
                  if (existing !=null && !existing.getBillingPeriod().hasRetailOwner()) {
                        existing.unregister(item);
                  }
                  if (existing !=null) {
                        existing.remove(item);
                  }
            }
      }
}
```

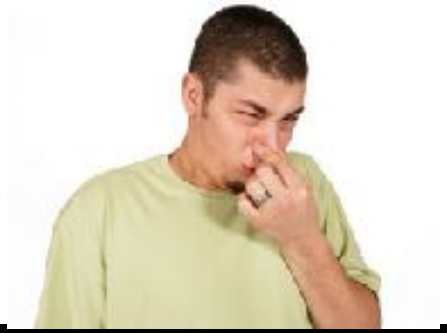
Don’t Return Null

If we change getEmployee so that it returns an empty list

```
List<Employee> employees = getEmployees();
for(Employee e : employees) {
        totalPay += e.getPay();
}
```

Fortunately, Java has Collections.emptyList(), and it returns a predefined immutable

```
public List<Employee> getEmployees() {
if( .. there are no employees .. )
        return Collections.emptyList();
}
```



```
public void registerItem(Item item) {
    if (item != null) {
        ItemRegistry registry =
        peristentStore.getItemRegistry();
        Item existing = registry.getItem(item.getID());
        if (existing.getBillingPeriod().hasRetailOwner()) {
                existing.register(item);
        }
    }
}
```

No nullable parameters

```
public void registerItem(Item item) {
    ItemRegistry registry = peristentStore.getItemRegistry();
    Item existing = registry.getItem(item.getID());
    if (existing.getBillingPeriod().hasRetailOwner()) {
                existing.register(item);
    }
}
```

- It's like a **`boolean`** ⇨ split that function in 2: one for null, one for not-null
- Thoroughly check parameters at boundaries (=**defensive programming**)

```
double getPayAmount() {
  double result;
  if (_isDead) result = deadAmount();
  else {
   //logic
    if (_isSeparated) result = separatedAmount();
    else {
      //logic
      if (_isRetired) result = retiredAmount();
      else result = normalPayAmount();
    };
  }
  return result;
```

Replace Nested Conditional with Guard Clauses

```
double getPayAmount() {
  if (_isDead) return deadAmount();
  if (_isSeparated) return separatedAmount();
  if (_isRetired) return retiredAmount();
  return normalPayAmount();
};
```

```
public List<Integer> stringsToInts(
                    List<String> strings) {
    if (strings != null) {
        List<Integer> integers = new ArrayList<>();
        for (String s : strings) {
            integers.add(Integer.parseInt(s));
        }
        return integers;
    } else {
        return null;
    }
}
```

use Early returns

```
public List<Integer> stringsToInts2(
                    List<String> strings) {
   if (strings == null) {
      return null;
   }
   List<Integer> integers = new ArrayList<>();
   for (String s : strings) {
      integers.add(Integer.parseInt(s));
   }
   return integers;
}
```

- Reduces the number of indentations

Remove nulls

```
public List<Integer> stringsToInts2(
                    List<String> strings) {

   List<Integer> integers = new ArrayList<>();
   for (String s : strings) {
      integers.add(Integer.parseInt(s));
   }
   return integers;
}
```

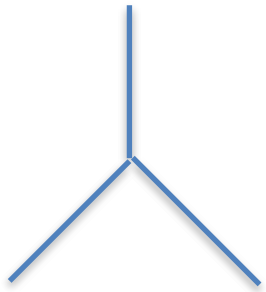
```
if (rowCount > rowIdx) {
      if(drc[rowIdx].Table.Columns.Contains("avalId")) {
      do {
            if (Attributes[attrVal.AttributeClassId] == null) {
            // do stuff
            }
            else {
                  if (!(Attributes[attrVal.AttributeClassId] is ArrayList)) {
                   // do stuff
                   }
                  else {
                        if (!isChecking) {
                        // do stuff
                        } else
                        {
                        // do stuff
                        }
                  }
            }
            rowIdx++;
      } while(rowIdx < rowCount && GetIdAsInt32(drc[rowIdx]) == Id);
}
else rowIdx++;
}
return rowIdx;
```

Cyclometic Complexity

```
public void ProcessPages()
{
 while(nextPage !=true)
 {
  if((lineCount<=linesPerPage) && (status != Status.Cancelled)
&& (morePages == true))
  {
   //....
  }
 }
}
```

Cyclometic Complexity

```
public int getValue(int param1)
{
 int value = 0;
 if (param1 == 0)
 {
  value = 4;
 }
 else
 {
  value = 0;
 }
 return value;
}
```



Cyclometic Complexity

CC Value	**Risk**
1-10	Low risk program
11-20	Moderate risk
21-50	High risk
>50	Most complex and highly unstable method

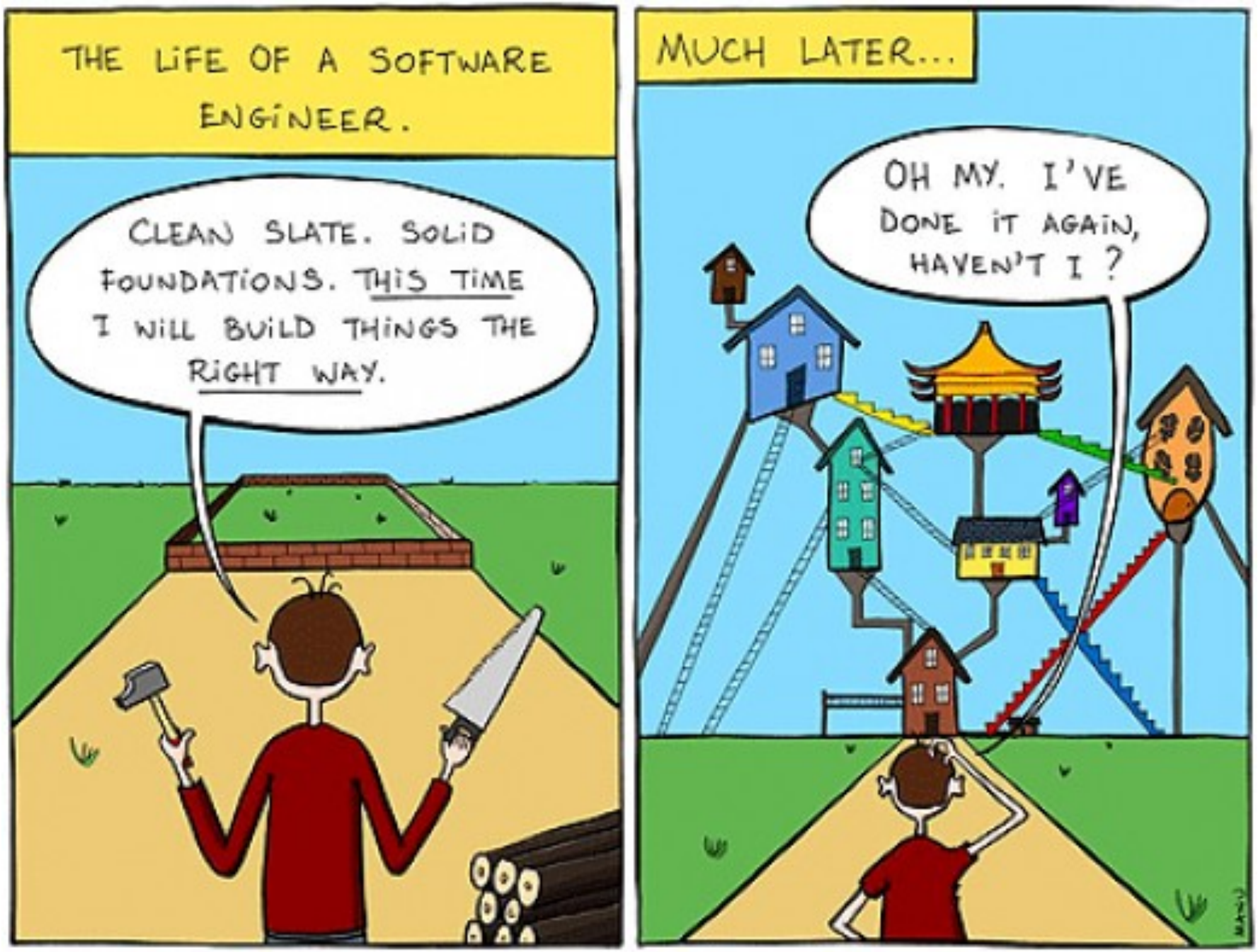
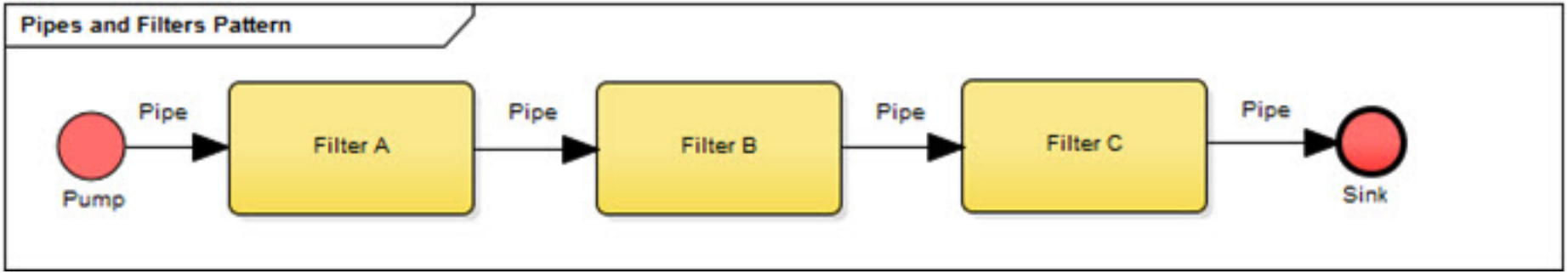
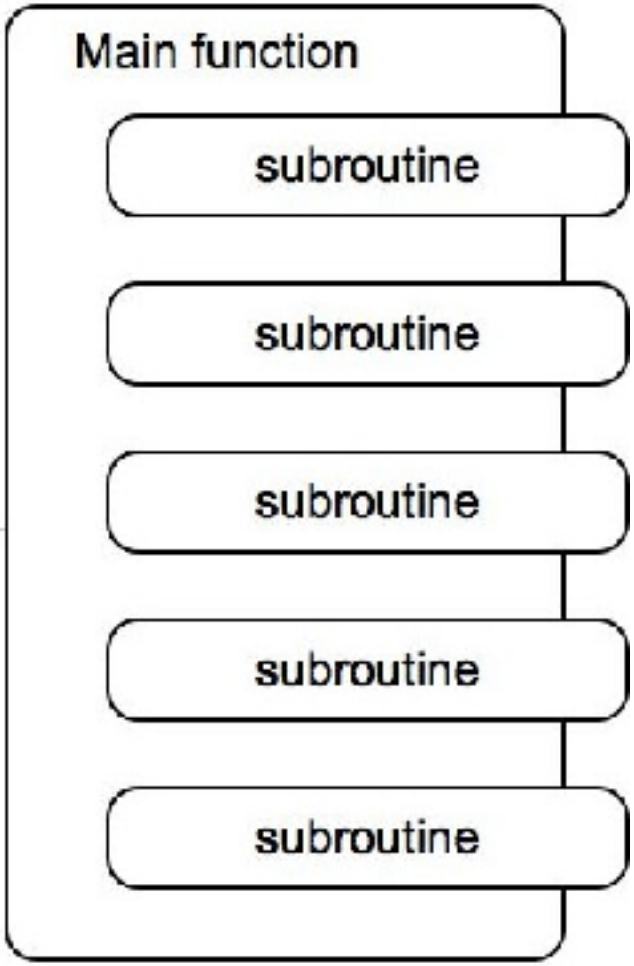

THE LIFE OF A SOFTWARE ENGINEER.
CLEAN SLATE. SOLID FOUNDATIONS. THIS TIME I WILL BUILD THINGS THE RIGHT WAY.
MUCH LATER...
OH MY. I'VE DONE IT AGAIN, HAVEN'T I ?

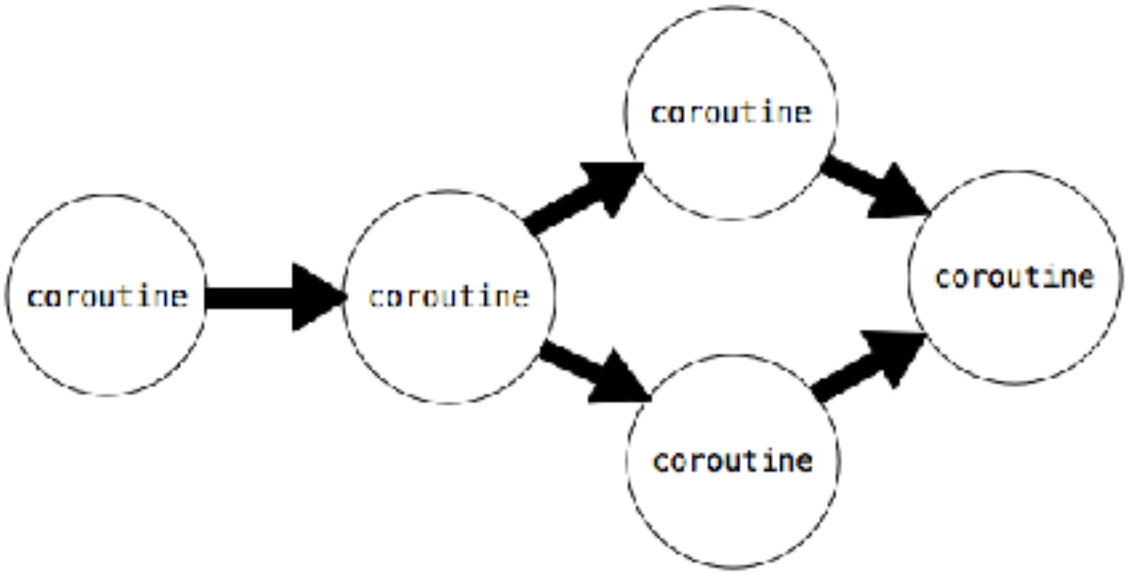
Image processing

- Capture Image
- Remove Noise in the image
- Identify Image type
- Identify Widgets in the image
- Mask sensitive widgets
- Extract text from widgets


Pipes and Filters Pattern
Pump
Pipe
Filter A
Pipe
Filter B
Pipe
Filter C
Pipe
Sink

Subroutine vs coroutine

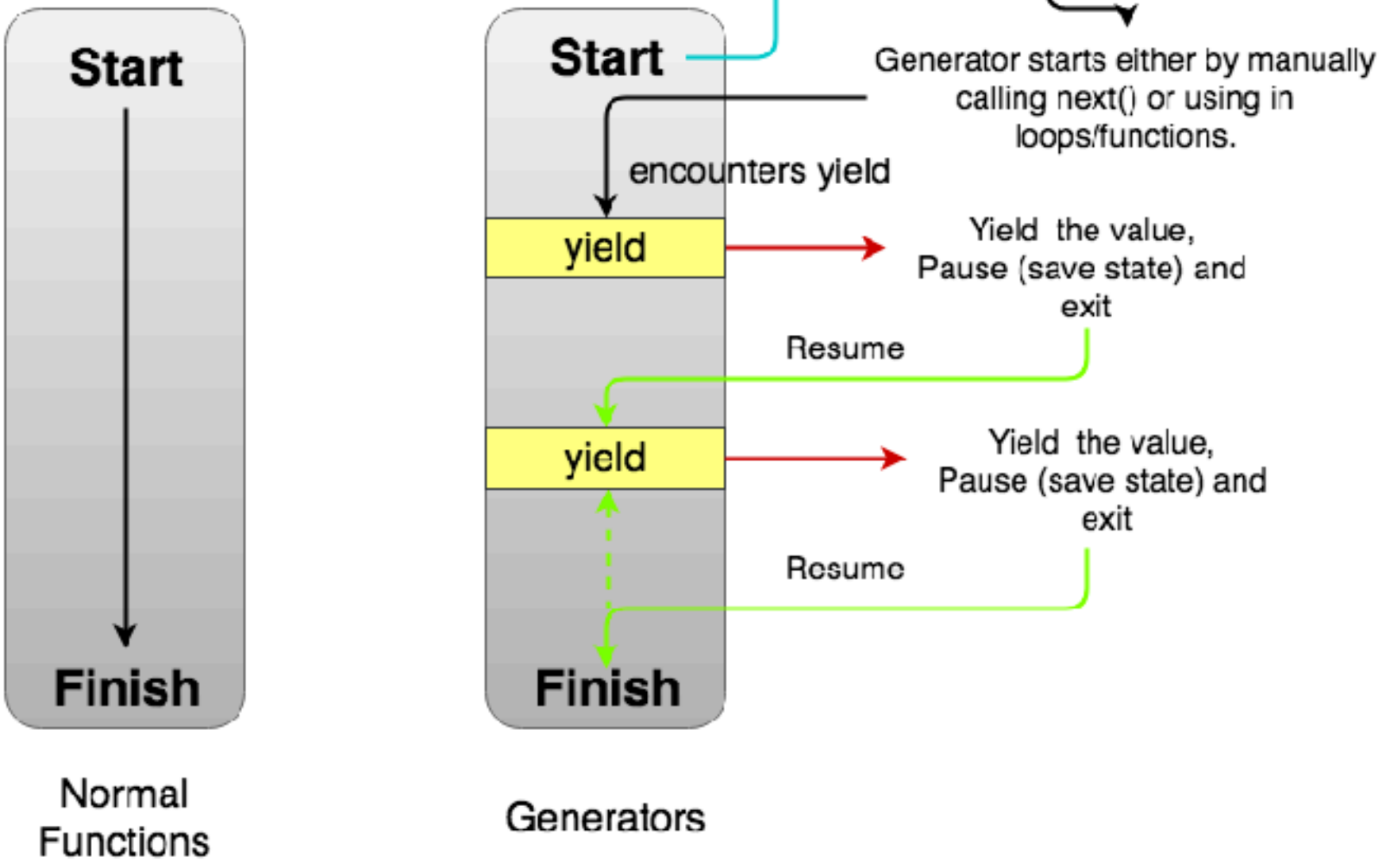




COroutine	*SUBroutine*
can be entered, exited, and resumed at many different points.	*invoked once and executes until it completed*
can pause execution and yield control back to the caller or another coroutine. The caller can then resume the coroutine when appropriate.	

```
def getNextWordGenerator():
    yield "Hello"
    yield "this"
    yield "is"
    yield "an"
    yield "example"

g = getNextWordGenerator()

print( g.__next__() )
print( g.__next__() )
print( g.__next__() )
```

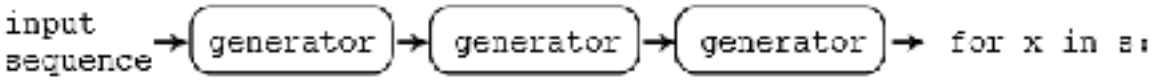
a generator is also an iterator


Start
Finish
Normal Functions
Start
yield
yield
Finish
Generators
returns a generator object (iterator).
Generator starts either by manually calling next() or using in loops/functions.
encounters yield
Yield the value, Pause (save state) and exit
Resume
Yield the value, Pause (save state) and exit
Resume

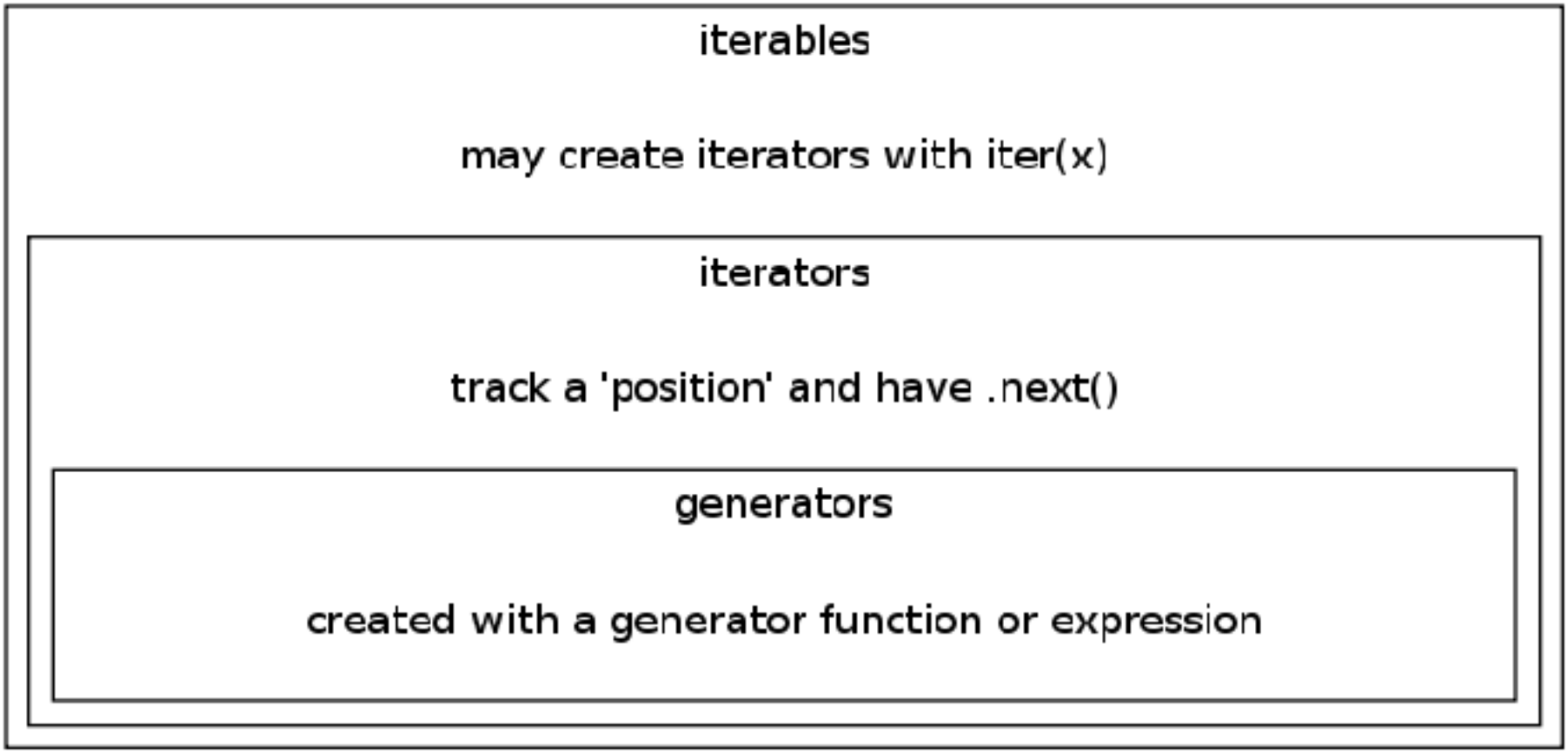
```
def MyIterator():
    print "Before First"
    yield "First"
    print "After First"

    for i in xrange(3):
        yield str(i)

    print "Before Last"
    yield "Last"
    print "After Last"

for s in MyIterator():
    print s
```

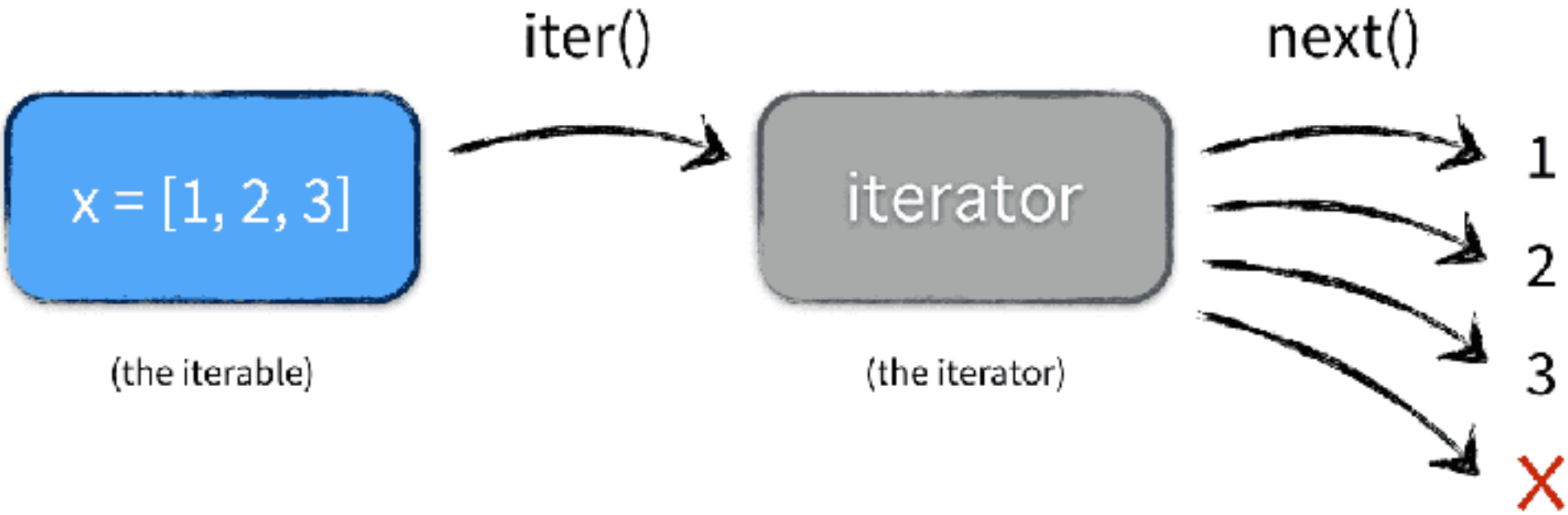
#**Generator Pipeline**

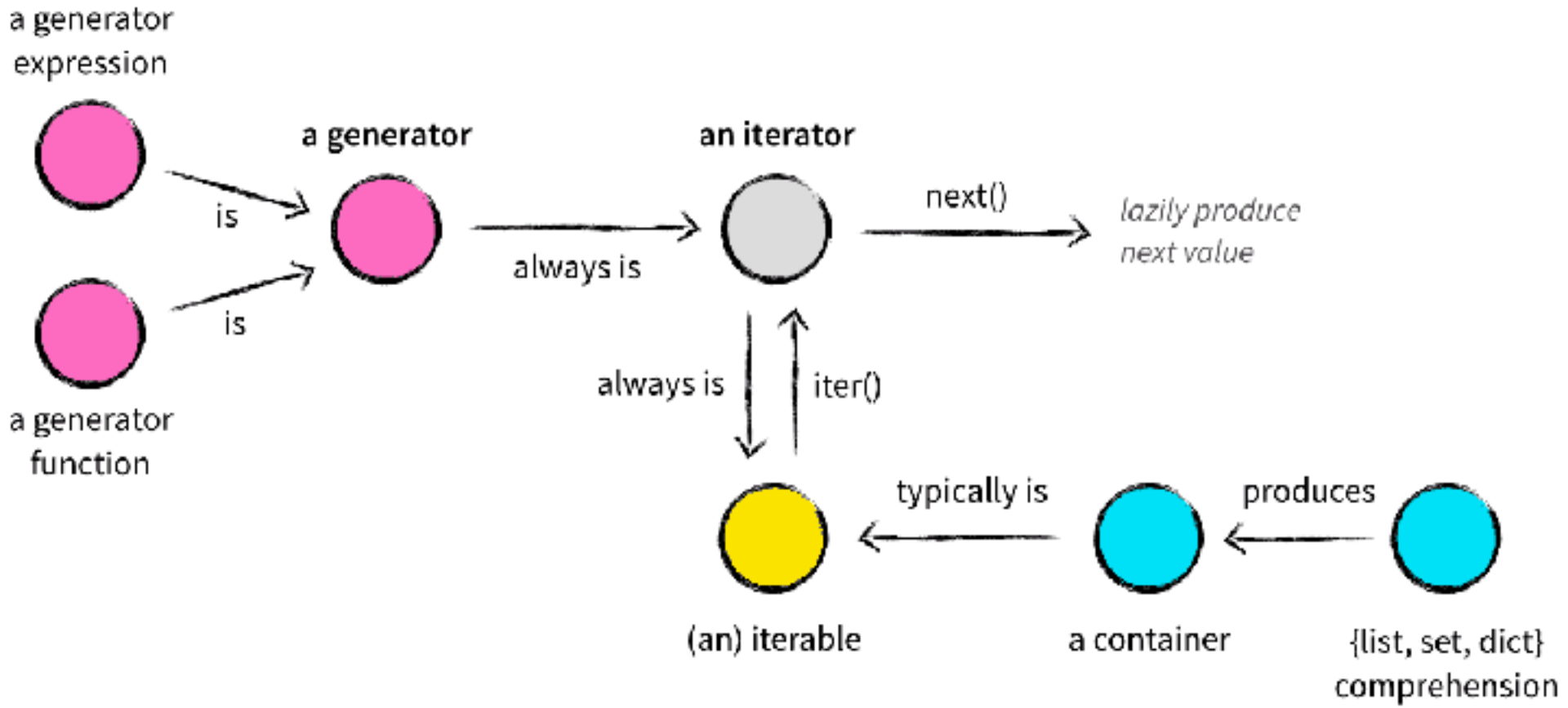
```
def even_filter(nums):
    for num in nums:
        if num % 2 == 0:
            yield num
```




```
def multiply_by_three(nums):
    for num in nums:
        yield num * 3

def convert_to_string(nums):
    for num in nums:
        yield 'The Number: %s' % num

nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
pipeline = convert_to_string(multiply_by_three(even_filter(nums)))
for num in pipeline:
    print (num)
```


iterables
may create iterators with iter(x)
iterators
track a 'position' and have .next()
generators
created with a generator function or expression


iter()
next()
x = [1, 2, 3]
iterator
1
2
3
X
(the iterable)
(the iterator)


a generator expression
is
a generator function
a generator
is
always is
an iterator
next()
lazily produce next value
always is
iter()
(an) iterable
typically is
a container
produces
{list, set, dict} comprehension

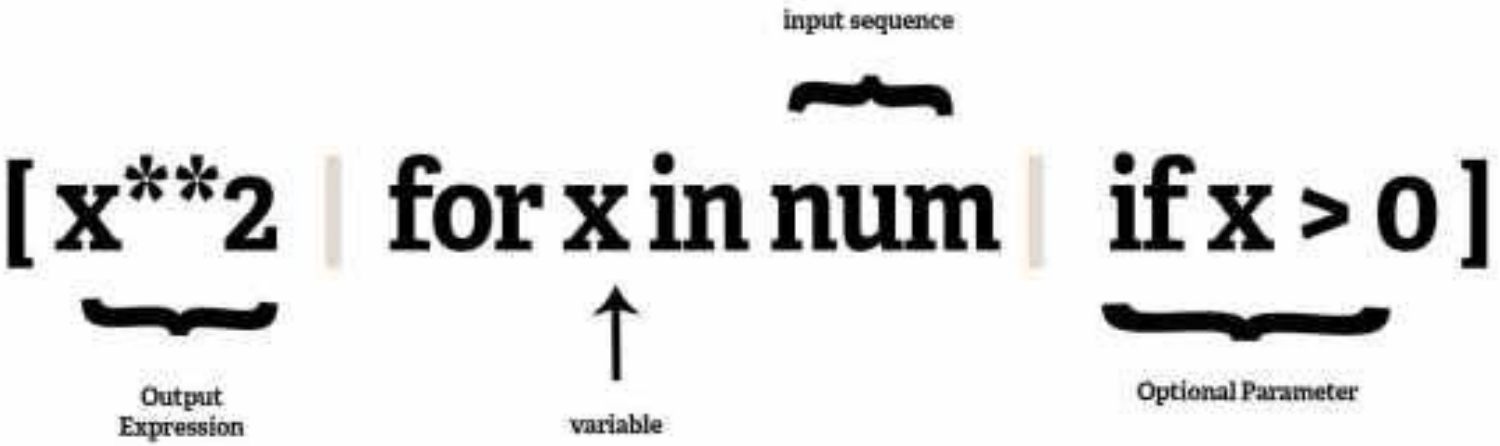
Imperative

```
let array = [0, 1, 2, 3, 4, 5]

var evenNumbers = [Int]()

for element in array {
 if element % 2 == 0 {
 evenNumbers.append(element)
 }
}
```

Declarative

```
let array = [0, 1, 2, 3, 4, 5]
let evenNumbers = array.filter { $0 % 2 == 0 }
```

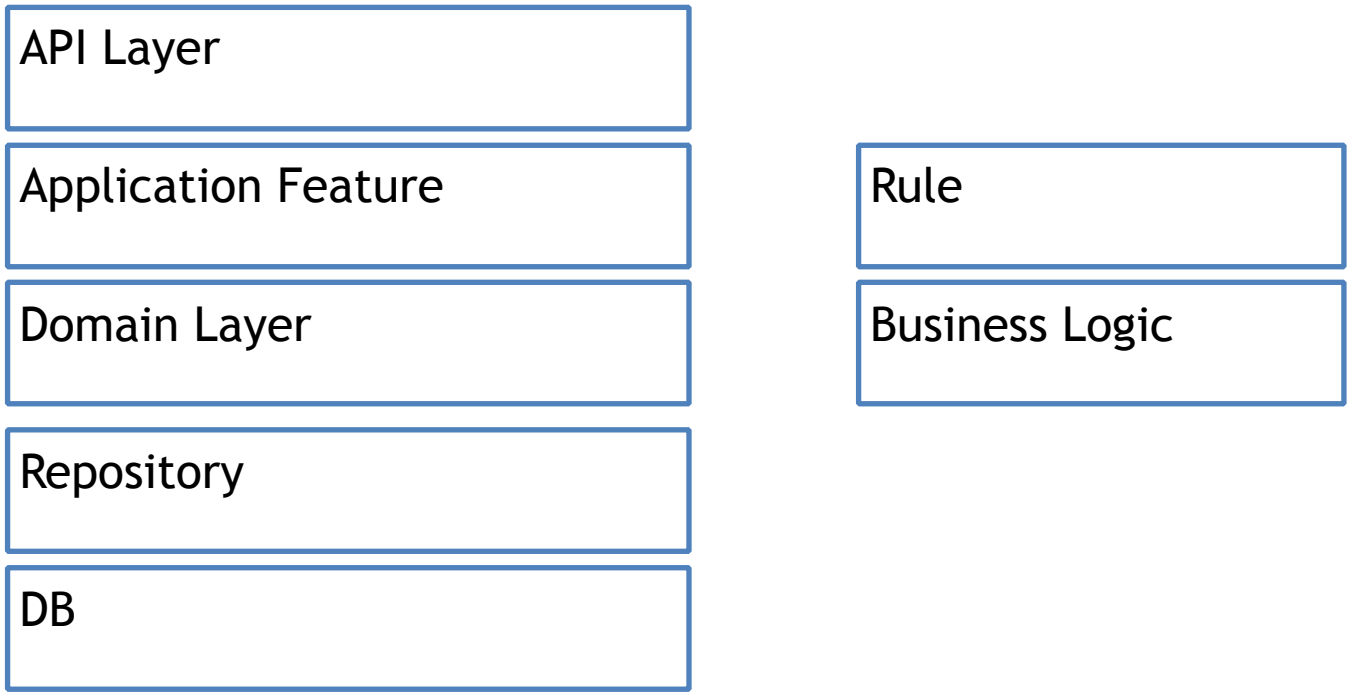
res = map(lambda x: x * 2 + 10, nums)



```
# approach 1 <- loop
for number in num:
    if number > 0:
        res.append(number ** 2)

# approach 2 <- map
res= map(lambda x: x ** 2, filter(lambda x: x > 0, num))

# approach 3 <- list comprehension
res = [ x**2 for x in num if x > 0]

# approach 4 <- generation expression
res = ( x**2 for x in num if x > 0 )
```

Lazy vs Eager

Use list comprehensions when the result needs to be iterated over multiple times, or where speed is paramount. Use generator expressions where the range is large or infinite.


API Layer
Application Feature
Domain Layer
Repository
DB
Rule
Business Logic