FUNCTION CACHING

```
IMMUTABLE
STABLE
VOLATILE
```

SECURITY CONTEXT

```
SECURITY DEFINER
```

CONTROL FLOW

```
if foo =='blah':
 do_something()
for x in somearray:
 statements
[x['somefield'] for x in rv]
[1.1*x/(2 + x) for x in range(5)]

if some boolean expression:
 stuff happens

while b < n:
   stuff happens
```

RETURN constructs

```
return somevariable
return somearray_variable
```

Common Constructs

```
import somepackage
#this is a comment
a, b = b, a+b
alist =['Jim', 'Guru', 'x3456']
name, title, phone = alist
alist = [] #declares empty array
adict = {} #declare empty dictionary
adict = {'Jim': 'Guru', 'Jimmy' : 'Intern'}
```

Constants and functions

```
True
False
coerce(somestring, ther)
dict(..)
globals()
float(num_string)
hasattr(object, name)
hash(obj)
int(num_string)
len(somearray_ordict)
long(num_string)
map(function, sequence[, sequence, ...])
pow(x, y [, z])
range([start,] end [, step])
xrange(start [, end [, step]]) (#use for big lists)
round(x,[numdigits])
slice([start,] stop[, step])
split(pattern,string, [maxsplit])
str(object)
zip(seq1[, seq2,...])
somestring.split
```

Operators

```
+ - *
/ // *
** # power operator
& ^ | == #boolean operators
&= ^= |= #compound bool
<< >> #shift operators
<<= >>= #compound shift operators
```

EXCEPTION Handling

```
try:
 stuff happens
 return something
except (IOError, OSError):
 return "an error has happened"

try:
 stuff happens
 return something
except IOError:
 return "an IOError"
except RuntimeError:
 return "runtime error"
except:
 return "have no clue what happened"
```

Common Error States

```
Exception
 StandardError
 ArithmeticError
  FloatingPointError
  OverflowError
  ZeroDivisionError
 EnvironmentError
  IOError
  OSError
 EOFError
 ImportError
 RuntimeError
 SystemError
```

Built-in Objects

```
plpy
 execute(sql) #returns a python dictionary object
 prepare(sql) # returns a prepared plan object
TD["new"] # trigger new data
TD["old"] # trigger old data
TD["when"] # BEFORE, AFTER, UNKNOWN
SD
```

Common Packages and Package Functions

```
os -- chdir(path), chmod(path), listdir(path)
 mkdir(path,[mode]), rmdir(path), unlink(path),
 write(fp, str), path.exists()
math -- pi
sys -- argv, modules,path,stdin,stdout,
 stderr, version, exit(n)
time -- time(), clock(), strftime(format, timetuple), sleep(secs)
```

COSTING

```
COST cost metric
ROWS estimated number of rows
```

Official PostgreSQL 8.3 PL/Python Documentation URL: http://www.postgresql.org/docs/8.3/interactive/plpython.html
Official Python documentation: http://docs.python.org/
We cover only a subset of what we feel are the most useful constructs that we could squash in a single cheatsheet page
commonly used
[1] New in this release.

PLPYTHON FUNCTION SAMPLES

```
CREATE OR REPLACE FUNCTION readfile (param_filepath text)
  RETURNS text
AS $$
 import os
 if not os.path.exists(param_filepath):
  return "file not found"
 return open(param_filepath).read()
$$ LANGUAGE plpythonu;

SELECT readfile('/test.htm');
```

```
--Doing postgresql queries --
CREATE OR REPLACE FUNCTION getmydata (param_numitems integer)
  RETURNS SETOF mydata
AS $$
 #lets us grab the first param_numitems records
 rv = plpy.execute("SELECT pt_id, pt_observation FROM mydata",param_numitems);
 return rv
$$ LANGUAGE plpythonu;

SELECT *
 FROM getmydata(20) As d;
```

```
--Example Using custom classes and local classes --
CREATE OR REPLACE FUNCTION dofunkyplot (param_anum integer)
  RETURNS text
AS $$
 import aplotter
 import sys
 class WritableObject:
        def __init__(self):
                self.content = ''
        def write(self, string):
                self.content = self.content + string
 saveout = sys.stdout
 outbuffer = WritableObject()
 sys.stdout = outbuffer
 #range (param_anum) will return array
 #consisting of 0 to param_num - 1 and formula
 # gets applied to each element
 # [1.1*x/(2 + x) for x in range(2)] -> [0 1.1/3]
 aplotter.plot([1.1*x/(2 + x) for x in range(param_anum)])
 sys.stdout = saveout
 return outbuffer.content
$$ LANGUAGE plpythonu;

SELECT dofunkyplot(n)
FROM generate_series(5,20, 10) As n;
```