Pyglish Documentation

Pyglish transpiles into readable python…

It provides syntax much closer to our everyday English and is supposed to help non-programmers write programs for their everyday needs much easier.

# Formats

Files ending with .pyg are treated as pyglish files, files must end with .pyg to be transpiled into python.

# Basic Features

## (Could be) strongly typed

Why is that useful? First and foremost, readability counts.

Secondly, where documentation is needed, the given type will be used in the documentation. This partially helps solve some problems of temporal coupling between code and documentation. (That is, when code changes but documentation remains)

### Variable Declaration

**Pyglish**

>>> int x = 5

**Python Equivalent**

>>> x = 5

### Fields

**Pyglish**

>>> class Foo(object):  
 """Foo class.   
 Attributes:  
 x: foo's x.   
 y: foo's y.  
 z (char): foo's z.  
 """  
 def \_\_init\_\_(self, int x, y, str z):  
 pass

**Python Equivalent**

>>> class Foo(object):  
 """Foo class.   
 Attributes:  
 x (int): foo's x.   
 y: foo's y.  
 z (char): foo's z.  
 """  
 def \_\_init\_\_(self, x, y, z):  
 pass

### Iteration

**Pyglish**

>>> for str name in names:  
 pass

**Python Equivalent**

>>> for name in names:  
 pass

### Arguments

**Pyglish**

>>> def bar(int x):  
 """Compute a bar.  
  
 Args:  
 x: x to compute.  
 """  
 pass

**Python Equivalent**

>>> def bar(x):  
 """Compute a bar.  
  
 Args:  
 x (int): x to compute.  
 """  
 pass

# Syntatic Sugar

### Loops

#### Until Loops

**Pyglish**

>>> until x == 10:  
 x += 1  
 print x

**Python Equivalent**

>>> while not (x == 10):  
 x += 1  
 print x

#### As Long As Loops

**Pyglish**

>>> as long as name is not None:  
 x += 1  
 print x

**Python Equivalent**

>>> while name is not None:  
 x += 1  
 print x

#### Times Loops

**Pyglish**

>>> 10 times:  
 print "Hello World"

**Python Equivalent**

>>> for \_ in xrange(10):  
 print "Hello World"

#### Repeat Times Loops

**Pyglish**

>>> repeat 10 times:  
 print "Hello World"

**Python Equivalent**

>>> for \_ in xrange(10):  
 print "Hello World"

#### Do-While Loops

**Pyglish**

>>> do:  
 x += 2  
 while x is not 10

**Python Equivalent**

>>> x += 2  
while x is not 10:  
 x += 2

**OR**

>>> while True:  
 x += 2  
if x is not 10:  
 break

#### foreach Loops

Trivial

#### for each Loops

Trivial

### Conditions

#### whether

>>> whether x is 10, 20, or 30:  
 pass

>>> if x is 10 or x is 20 or x is 30:  
 pass

>>> whether not\_logged\_in(), timed\_out() or error\_recieved():  
 pass

>>> if not\_logged\_in() or not timed\_out() or error\_recieved():  
 pass

#### either

>>> if either not\_logged\_in(), timed\_out() or 1==3: # But not both  
 pass

>>> if bool(not\_logged\_in()) ^ bool(timed\_out()) ^ bool(1==3):  
 pass

>>> if x is either 1, 2, 3, or 4?  
 pass

>>> if bool(x is 1) ^ bool(x is 2) ^ bool(x is 3 ) ^ bool(x is 4):  
 pass

#### if then otherwise

>>> if key is None then prompt\_admin() otherwise prompt\_user()

>>> if key is None:  
 prompt\_admin()  
 else:  
 prompt\_user()

#### neither

>>>if error is neither RuntimeError, DivisionByZeroError nor socket.error:  
 pass

>>> if error is not RuntimeError and error is not DivisionByZeroError and error is not socket.error:  
 pass

>>> if neither message\_being\_sent() nor user\_is\_typing():  
 pass

>>> if not message\_being\_sent() and not user\_is\_typing():  
 pass

#### Multiple is statements

>>>if client is self.\_server, self.\_admin or None:   
 pass

>>> if client is self.\_server or client is self.\_admin or client is None:  
 pass

>>>if id is 6 or 5:   
 pass

>>> if id is 6 or id is 5:  
 pass

However, one might ask, what happens with the following on Pyglish:  
>>> if number is 6 or some\_other\_condition():  
 pass

What if we want to preserve this pythonic meaning?  
  
So a couple of tricks one might do   
1. >>> if number is 6 or some\_other\_condition()==True:  
 pass  
2. >>> if (number is 6) or some\_other\_condition():  
 pass  
3. >>> # pyglish: disable=multiple-is  
 if number is 6 or some\_other\_condition():  
 pass

#### is in

>>>if RuntimeError is in errors:  
 pass

>>> if RuntimeError in errors:  
 pass

#### contains

>>>if words\_list contains "dog":  
 pass

>>> if "dog" in words\_list:  
 pass

# Assignment Return Value

Assigning a variable return it's value, similar to C language. We find this very similar to the way we do our everyday mathematics.

## Casual Assignment

>>> a=b=c=0

>>> c =0  
 b=c  
 a=b

## Conditional Assignment

>>> if a=b:  
 pass

This one provoked some thought from our team.  
Which is more readable, when *if (a=b)* means **if(a==b)**, or **a=b; if (a)**?  
Surprisingly or not, we chose the latter.

>>> if a==b:  
 pass

# Unit Testing

You'll import *unittest* and call your methods test\_ and your classes will inherit *TestCase*, however, the assertion part will be made more pyglish.

## Basic

**>>> assert that x == y**

**>>> self.assertTrue(x==y)**

**>>> assert that x is equal to y**

**>>> self.assertEquals(x,y)**

>>> assert that x is equal to y, or print "X was not equal to Y!"

>>> self.assertEquals(x, y, "X was not equal to Y!")

**>>> assert that x is not equal to y**

**>>> self.assertNotEquals(x, y)**

**>>> assert that x is greater than or equal to y**

**>>> self.assertTrue(x >= y)**

>>> fail if x > y

>>> if x > y: self.fail()

**>>> assert that x is less than y**

**>>> self.assertTrue(x < y)**

**>>> assert that expected is actual**

**>>> self.assertTrue(expected is actual)**

>>> assert that a contains b

>>> self.assertIn(b, a)

>>> assert that b is in a

>>> self.assertIn(b, a)

>>> assert that foo raises Exception

>>> self.assertRaises(foo, Exception)

## Complex Conditions

>>> assert that a contains b or b contains a

>>> self.assertTrue(b in a or a in b)

>>> assert that either a contains b or b contains a

>>> self.assertTrue(b in a ^ a in b)

## Advanced Assertions

>>> assert that list(1, 2) is not empty

>>> self.assertTrue(len(list(1, 2)) != 0)

>>> assert that x is iterable

>>> import collections; self.assertTrue(isinstance(x, collections.Iterable))

>>> sub = (1, 2, 3); assert that x contains sub in order

>>> self.assertTrue(any(x[i:i+len(sub)]==sub for i in xrange(len(x)-len(sub)+1))

>>> assert that x is close to y with offset 10

>>> self.assertTrue(abs(x-y) <= 10)

# TODOS:

Exit codes

Let

Var

File i/o

Socket i/o

Threading wait

Basic math

Basic logging

Datetime.datetime.now()

@before\_var\_name

Strangle

Abstract class

A=b=3

run

switch case

binary,

0x

#pyglish ignore

Upon keyword – listening for event by metaprogamming

Then keyword

Success keyword

Failure keyword

As keyword monoiker