Latest stable release: 3.7.8 'London' - monthly downloads: 13,286

Clear standard syntax

Transcrypt has exactly the same clear, powerful syntax that Python is famous for, without the need for any proprietary extensions.

It supports string slicing with [:|:k], matrix and vector operations with +, -, *, / and more, out of the box.

It precompiles to fast, readable JavaScript that can be debugged from the Python source code using sourcemaps.

Superior scalability

Python was designed for large scale programs from the ground up.

Hierarchical modules, local classes and multiple inheritance are all supported by Transcrypt, allowing a flexible, yet stable overall structure.

Transcrypt comes integrated with a static type validator, a linter and a minifier, enabling effective cooperation of large teams on extensive projects.

One project, one language

Python is used everywhere at the back-end, from web servers to scientific computing. Now you can use it at the front-end as well.

Transcrypt offers seamless access to any JavaScript library and also runs on top of Node.js.

Python source code and JavaScript target code roughly have the same size, so your pages load as fast as ever.

Get started: Hello, solar system...

1. Download Python 3.7 from www.python.org
2. Install Transcrypt from the command prompt by typing *python -m pip install transcrypt*
3. Create a new folder *hello* containing *hello.html* and *hello.py*
4. Go to that new folder and type *python -m transcrypt -b -m -n hello.py*
5. In that same new folder start an HTTP server by typing *python -m http.server*
6. In your browser, navigate to *localhost:8000/hello.html* to see the result

- On Windows systems you may have to type *python37* or *python3* rather than *python*
- On Linux and MacOS systems you may have to type *python3.7* or *python3* rather than *python*
- On some systems you can just type *transcrypt -b -m -n hello.py* to compile the test program

Run the 'hello' example

Code in hello/hello.html:
```
<script type="module">imp
```

Code in hello/hello.py:
```
from itertools import chain
```

If you have trouble installing Transcrypt or compiling this example, you'll find additional information in the getting started chapter of the documentation.

Harness the power of Python for increasingly complex web applications

Anything you can do in JavaScript, you can do in Transcrypt. You'll always have immediate access to the newest JavaScript libraries, use the newest DOM functions, interact with the newest HTML or CSS. Calling JavaScript functions from Python, embedding JavaScript code in your Python source, attaching Python event handlers to DOM elements, passing native data between Python and JavaScript, it all happens without any conversion or special syntax. Just join the party without restrictions.

Transcrypt is precompiled for speed, rather than interpreted in the browser. It produces small, fast, yet readable code. The minified downloads are measured in kB's rather than MB's.

Transcrypt stays as close to the Python original as possible without sacrificing performance. Multiple inheritance, recursive tuple assignment, multi-loop nested list comprehensions, LHS and RHS extended slices, assignment of bound functions, lambdas, named, default, *args and **kwargs parameters, properties, optional operator overloading, iterators, generators, async/await, selective exception handling and a hierarchical module system are just a few of its characteristics that make this clear. Transcrypt is parsed by CPython's AST module, so no surprises there.

Transcrypt features multi-level sourcemaps, so you can debug your application comfortably, working from the Python source code in your browser, even if the generated JavaScript is minified. It also has a unique autotest feature that makes back to back regression testing with CPython a snap.

Transcrypt comes with 3rd party tools for optional static type validation, lightweight consistency checks and minification, all at the tip of a command line switch. It generates code for JavaScript 6. All tools are fully integrated in the distribution, one single pip install will put them at your disposal.

Even with multiple inheritance there's a simple correspondence between Python and JavaScript

The ability to understand exactly what's going on under the hood, allows for fine grained optimizations. Speed up an inner loop? Use *__pragma__ ('js', ...)* to freely mix Python with native JavaScript. Not that you'll need it often... An expression like *i++1* is compiled to *i++*, and optional call caching bypasses the prototype chain, making repeated function calls even faster than handwritten JavaScript.

classes.py
```
class A:
    def __init__ (self, x):
        self.x = x

    def show (self, label):
        print ('A.show', label, self.x)

class B:
    def __init__ (self, y):
        alert ('In B constructor')
        self.y = y

    def show (self, label):
        print ('B.show', label, self.y)

class C (A, B):
    def __init__ (self, x, y):
        alert ('In C constructor')
        A.__init__ (self, x)
        B.__init__ (self, y)
        self.show ('constructor')

    def show (self, label):
        B.show (self, label)
        print ('C.show', label, self.x, self.y)

a = A (1001)
a.show ('america')

b = B (2002)
b.show ('russia')

c = C (3003, 4004)
c.show ('netherlands')

show2 = c.show
show2 ('copy')
```

classes.js
```
var A = __class__ ('A', [object], {
    get __init__ () {return __get__ (this, function (self, x) {
        self.x = x;
    });},
    get show () {return __get__ (this, function (self, label) {
        print ('A.show', label, self.x);
    });}
});
var B = __class__ ('B', [object], {
    get __init__ () {return __get__ (this, function (self, y) {
        alert ('In B constructor');
        self.y = y;
    });},
    get show () {return __get__ (this, function (self, label) {
        print ('B.show', label, self.y);
    });}
});
var C = __class__ ('C', [A, B], {
    get __init__ () {return __get__ (this, function (self, x, y) {
        alert ('In C constructor');
        A.__init__ (self, x);
        B.__init__ (self, y);
        self.show ('constructor');
    });},
    get show () {return __get__ (this, function (self, label) {
        B.show (self, label);
        print ('C.show', label, self.x, self.y);
    });}
});
var a = A (1001);
a.show ('america');
var b = B (2002);
b.show ('russia');
var c = C (3003, 4004);
c.show ('netherlands');
var show2 = c.show;
show2 ('copy');
```

Transcrypt is suitable for a broad area of applications

Transcrypt supports customization through the use of *pragmas*. Pragmas are function calls that allow you to switch on and off compiler facilities locally, to conditionally compile code, comparable to the use of *#ifdef...#endif* and *#ifndef...#endif* in C++ and to include native JavaScript code anywhere in your program. They also allow efficient, selective operator overloading.

Many people know Python from internet programming, especially from the Django webserver. But it is also tremendously popular in world of scientific computing and education. Selective operator overloading enables the use of elegant notation for mathematical expressions, making Transcrypt an attractive choice for writing small to medium scale numerical applications that have to run in a browser.

Being able to write matrix multiplication and vector addition as

```
v3 = M3 * (M1 * v1 + M2 * v2)
```

rather than

```
v3 = multiply (M3, add (multiply (M1, v1), multiply (M2, v2)))
```

is a big advantage if the formulas get complicated.

Being able to write

```
c = (3 + 5j) * (-4 - 2j) + (1 + 1j)
```

rather than

```
c = add (multiply (complex (3, 5), complex (-4, -2)), complex (1, 1)
```

when complex numbers are involved is equally handy.

While operator overloading is very powerful, when compiling to JavaScript it is important to be able to switch it on and off selectively to prevent something simple like

```
x = 3 * 4 + 5 * 6
```

from being translated to

```
x = add (multiply (3, 4), multiply (5, 6)
```

everywhere in your program.

An example of the efficient use of operator overloading is Numscrypt. Numscrypt is a port of a small part of Numpy to Transcrypt. It currently supports real and complex matrix and vector multiplication, division, addition, subtraction, dot product, inverse, transpose, FFT, IFFT, FFT2 and IFFT2, using Numpy's familiar notation, including array slicing. To achieve speed, Numscrypt uses JavaScript TypedArray's under the hood. While it's just as fast and compact as comparable native JavaScript numerical libraries, its notation much more simple and concise, closely resembling mathematical notation.

It's free. It's open. It's yours.