Index

jQuery

Despite competition, jQuery is still ubiquitous. Transcrypt and jQuery wear well together.

Run the 'jquery_demo' example

Code in jquery_demo/jquery_demo.html:

```
<html>
    <head>
        <script
        src="https://ajax.googleapi
        s.com/ajax/libs/jquery/1.12
        .0/jquery.min.js"></script>
```

```
            <script type="module">
                import * as jquery_demo
                from
                "./__target__/jquery_de
                mo.js";
                $ (document) .ready
                (jquery_demo.start)
            </script>
        </head>
        <body bgcolor="black">
            <font face="arial" size =
            "8">
            <div>The</div>
            <div>quick</div>
            <div>brown</div>
            <div>fox</div>
            <div>jumps</div>
            <div>over</div>
            <div>the</div>
            <div>lazy</div>
            <div>dog</div>
        </body>
    </html>
```

Code in jquery_demo/jquery_demo.py:

```
    __pragma__ ('alias', 'S', '$')

    def start ():
        def changeColors ():
            for div in S__divs:
```

```
                S (div) .css ({
                    'color': 'rgb({},
                    {},{})'.format (*
                    [int (256 *
                    Math.random ()) for
                    i in range (3)]),
                })

        S__divs = S ('div')
        changeColors ()
        window.setInterval
        (changeColors, 500)
```

Back to index

React

The React library makes it possible to create fast reacting interactive GUI's, by modifying a virtual DOM first and then adapting the real DOM in a minimal way. This example shows how it can be used in combination with Transcrypt.

Run the 'react_demo' example

Code in react_demo/react_demo.html:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
```

```
        <title>Hello React!</title>
        <script
        src="https://unpkg.com/react@16
        /umd/react.development.js"
        crossorigin></script>
        <script
        src="https://unpkg.com/react-
        dom@16/umd/react-
        dom.development.js"
        crossorigin></script>
        <style>
            body {font-
            family:arial;font-
            size:30px;padding:50px;back
            ground-color:#eeeeee;}
            h1 {font-
            size:50px;color:#0000ff;}
        </style>
      </head>
      <body>
        <div id="container"></div>
        <script type="module">import *
        as react_demo from
        "./__target__/react_demo.js";
        </script>
      </body>
    </html>
```

Code in react_demo/react_demo.py:

```
    from org.reactjs import
    createElement, useState, useEffect,
```

```
    useRef
    from org.reactjs.dom import render
    as react_render


    # Helper functions

    def h(elm_type, props='', *args):
        return createElement(elm_type,
        props, *args)


    def render(react_element,
    destination_id, callback=lambda:
    None):
        container =
        document.getElementById(destina
        tion_id)
        react_render(react_element,
        container, callback)


    def useInterval(func, delay=None):
        # can be used as
        `useInterval(func, delay)`
        # or as `@useInterval(delay)`
        if delay is None:
            delay = func
            return lambda fn:
            useInterval(fn, delay)

        ref = useRef(func)
        ref.current = func
```

```
        @useEffect.withDeps(delay)
        def setup():
            id = setInterval(lambda:
            ref.current(), delay)
            return lambda:
            clearInterval(id)

        return func


    # Create a component

    def Hello(props):
        count, setCount = useState(0)

        @useInterval(1000)
        def updateCounter():
            setCount(count+1)

        return h(
            'div',
            {'className': 'maindiv'},
            h('h1', None, 'Hello ',
            props['name']),
            h('p', None, 'Lorem ipsum
            dolor sit ame.'),
            h('p', None, 'Counter: ',
            count),
            h(
                'button',
                {'onClick':
                updateCounter},
```

```
                'Increment',
            )
        )


    # Render the component in a
    'container' div

    element =
    React.createElement(Hello, {'name':
    'React!'})
    render(element, 'container')
```

Back to index

Pong: Multiple classes and encapsulating Fabric.js

This example illustrates the clean application structure that can achieved by using class based object orientation in combination with encapsulating a JavaScript library as a Python module. Encapsulating JavaScript libraries is never necessary and in many cases it's overkill, all libraries can be used as-is. Moreover in this example the use of the voluminous Fabric.js library, encapsulated or not, is in itself overkill as well. The same functionality could have been achieved easily directly on top of the HTML5 canvas. But it serves to illustrate the principle.

A case where such encapsulation on the contrary is very useful, is the implementation of the Fast Fourier Transform in Numscrypt. The API is modeled after Numpy, utilizing an *ndarray* of type *complex*. Under the hood an efficient JavaScript open source library is used. By marrying the two, this functionality smoothly fits into Numscrypt, complete with operator overloading and slicing. This case was less suitable as an example here, since it requires mathematical knowledge.

Run the 'pong' example

Code in pong/pong.html:

```
<html>
    <body>
        <font face="arial" >

        <style>
            #A, #Z, #K, #M, #space,
            #enter {
                -webkit-user-
                select:none;
                -khtml-user-
                drag:none;
                -khtml-user-
                select:none;
                -moz-user-
                select:none;
                -moz-user-select:-
                moz-none;
                -ms-user-
                select:none;
                user-select:none
            }
            body {
                visibility: hidden;
            }
        </style>

        <div id='text_frame'
        style="position:absolute">
```

```
                <font color="444444"
                size="3">
                <h3>Hit the spacebar to
                start</h3><br>

                <font color="000000"
                size="3">
                This example has
                sourcemap support for
                Google Chrome. Press
                F12, click the
                <i>sources</i> tab and
                then click <b>somewhere
                near the end</b> of
                <i>pong.js</i> to try.
                <br><br>

                You have to click near
                the end since 90% of
                the JS code is the
                large fabric.js
                library. Fabric.js is
                used here just as an
                example, this simple
                demo could have been
                programmed directly on
                top of the HTML5
                canvas.
            </div>

            <div id="canvas_frame"
            style="position:absolute">
```

```
            <canvas id="canvas">
            </canvas>
        </div>

        <div id="buttons_frame"
        style="position:absolute;
        font-size:30">
            <div id='A'
            style="position:absolut
            e; background-color:
            gray; left:0; top:0;
            width:100; height:90;
            text-align:center">
            <br>A</div>
            <div id='Z'
            style="position:absolut
            e; background-color:
            gray; left:0; top:120;
            width:100; height:90;
            text-align:center">
            <br>Z</div>

            <div id='K'
            style="position:absolut
            e; background-color:
            gray; left:400; top:0;
            width:100; height:90;
            text-align:center">
            <br>K</div>
            <div id='M'
            style="position:absolut
            e; background-color:
            gray; left:400;
```

```
                top:120; width:100;
                height:90; text-
                align:center">
                <br>M</div>

                <div id='space'
                style="position:absolut
                e; background-color:
                gray; left:0; top:240;
                width:270; height:90;
                text-align:center">
                <br>Space</div>
                <div id='enter'
                style="position:absolut
                e; background-color:
                gray; left:350;
                top:240; width:150;
                height:90; text-
                align:center">
                <br>Enter</div>

                <div
                style="position:absolut
                e; top:300"> 
                </div>
            <div>

            <script
            type="module">import * as
            pong from
            "./__target__/pong.js";
            window.pong = pong;
            </script>
```

```
            <style>body {visibility:
            visible;}</style>
        </body>
    </html>
```

Code in pong/pong.py:

```
__pragma__ ('skip')
document = window = Math = Date = 0
# Prevent complaints by optional
static checker
__pragma__ ('noskip')

__pragma__ ('noalias', 'clear')

from com.fabricjs import fabric

orthoWidth = 1000
orthoHeight = 750
fieldHeight = 650

enter, esc, space = 13, 27, 32

window.onkeydown = lambda event:
event.keyCode != space # Prevent
scrolldown on spacebar press

class Attribute:    # Attribute in
the gaming sense of the word,
rather than of an object
    def __init__ (self, game):
```

```
            self.game = game
                    # Attribute knows
            game it's part of
            self.game.attributes.append
            (self)  # Game knows all
            its attributes
            self.install ()
                    # Put in place
            graphical representation of
            attribute
            self.reset ()
                    # Reset attribute
            to start position

        def reset (self):       #
        Restore starting positions or
        score, then commit to fabric
            self.commit ()      #
            Nothing to restore for the
            Attribute base class

        def predict (self):
            pass

        def interact (self):
            pass

        def commit (self):
            pass

    class Sprite (Attribute):   # Here,
    a sprite is an attribute that can
    move
```

```
        def __init__ (self, game,
        width, height):
            self.width = width
            self.height = height
            Attribute.__init__ (self,
            game)

        def install (self):     # The
        sprite holds an image that
        fabric can display
            self.image = __new__
            (fabric.Rect ({
                'width':
                self.game.scaleX
                (self.width), 'height':
                self.game.scaleY
                (self.height),
                'originX': 'center',
                'originY': 'center',
                'fill': 'white'
            }))

        __pragma__ ('kwargs')
        def reset (self, vX = 0, vY =
        0, x = 0, y = 0):
            self.vX = vX        # Speed
            self.vY = vY

            self.x = x          #
            Predicted position, can be
            commit, no bouncing
            initially
            self.y = y
```

```
            Attribute.reset (self)
        __pragma__ ('nokwargs')

        def predict (self):     #
        Predict position, do not yet
        commit, bouncing may alter it
            self.x += self.vX *
            self.game.deltaT
            self.y += self.vY *
            self.game.deltaT

        def commit (self):      #
        Update fabric image for asynch
        draw
            self.image.left =
            self.game.orthoX (self.x)
            self.image.top =
            self.game.orthoY (self.y)

        def draw (self):
            self.game.canvas.add
            (self.image)

    class Paddle (Sprite):
        margin = 30 # Distance of
        paddles from walls
        width = 10
        height = 100
        speed = 400 # / s

        def __init__ (self, game,
        index):
```

```
        self.index = index  #
        Paddle knows its player
        index, 0 == left, 1 ==
        right
        Sprite.__init__ (self,
        game, self.width,
        self.height)

    def reset (self):       # Put
    paddle in rest position,
    dependent on player index
        Sprite.reset (
            self,
            x = orthoWidth // 2 -
            self.margin if
            self.index else -
            orthoWidth // 2 +
            self.margin,
            y = 0
        )

    def predict (self): # Let
    paddle react on keys
        self.vY = 0

        if self.index:
                        # Right player
            if self.game.keyCode ==
            ord ('K'):  # Letter K
            pressed
                self.vY =
                self.speed
```

```
                elif self.game.keyCode
                == ord ('M'):
                    self.vY = -
                    self.speed
            else:
                        # Left player
                if self.game.keyCode ==
                ord ('A'):
                    self.vY =
                    self.speed
                elif self.game.keyCode
                == ord ('Z'):
                    self.vY = -
                    self.speed

            Sprite.predict (self)
                        # Do not yet
            commit, paddle may bounce
            with walls

        def interact (self):    #
        Paddles and ball assumed
        infinitely thin
            # Paddle touches wall
            self.y = Math.max
            (self.height // 2 -
            fieldHeight // 2, Math.min
            (self.y, fieldHeight // 2 -
            self.height // 2))

            # Paddle hits ball
            if (
```

```
                (self.y - self.height
                // 2) <
                self.game.ball.y <
                (self.y + self.height
                // 2)
                and (
                    (self.index == 0
                    and
                    self.game.ball.x <
                    self.x) # On or
                    behind left paddle
                    or
                    (self.index == 1
                    and
                    self.game.ball.x >
                    self.x) # On or
                    behind right paddle
                )
            ):
                self.game.ball.x =
                self.x                  #
                Ball may have gone too
                far already
                self.game.ball.vX = -
                self.game.ball.vX  #
                Bounce on paddle
                self.game.ball.speedUp
                (self)

class Ball (Sprite):
    side = 8
    speed = 300 # / s
```

```
        def __init__ (self, game):
            Sprite.__init__ (self,
            game, self.side, self.side)

        def reset (self):   # Launch
        according to service direction
        with random angle offset from
        horizontal
            angle =  (
                self.game.serviceIndex
                * Math.PI    # Service
                direction
                +
                (1 if Math.random () >
                0.5 else -1) *
                Math.random () *
                Math.atan (fieldHeight
                / orthoWidth)
            )

            Sprite.reset (
                self,
                vX = self.speed *
                Math.cos (angle),
                vY = self.speed *
                Math.sin (angle)
            )

        def predict (self):
            Sprite.predict (self)
                # Integrate velocity to
            position
```

```
            if self.x < -orthoWidth //
            2:   # If out on left side
                self.game.scored (1)
                     #   Right player
                scored
            elif self.x > orthoWidth //
            2:
                self.game.scored (0)

            if self.y > fieldHeight //
            2:   # If it hits top wall
                self.y = fieldHeight //
                2   #   It may have
                gone too far already
                self.vY = -self.vY
                     #   Bounce
            elif self.y < -fieldHeight
            // 2:
                self.y = -fieldHeight
                // 2
                self.vY = -self.vY

        def speedUp (self, bat):
            factor = 1 + 0.15 * (1 -
            Math.abs (self.y - bat.y) /
            (bat.height // 2)) ** 2
            # Speed will increase more
            if paddle hit near centre

            if Math.abs (self.vX) < 3 *
            self.speed:
                self.vX *= factor
```

```
                self.vY *= factor


    class Scoreboard (Attribute):
        nameShift = 75
        hintShift = 25

        def install (self): # Graphical
        representation of scoreboard
        are four labels and a separator
        line
            self.playerLabels =
            [__new__ (fabric.Text
            ('Player {}'.format (name),
            {
                    'fill': 'white',
                    'fontFamily':
                    'arial',
                    'fontSize': '{}'
                    .format
                    (self.game.canvas.w
                    idth / 30),
                    'left':
                    self.game.orthoX
                    (position *
                    orthoWidth), 'top':
                    self.game.orthoY
                    (fieldHeight // 2 +
                    self.nameShift)
            })) for name, position in
            (('AZ keys:', -7/16), ('KM
            keys:', 1/16))]
```

```
            self.hintLabel = __new__
            (fabric.Text ('[spacebar]
            starts game, [enter] resets
            score', {
                    'fill': 'white',
                    'fontFamily':
                    'arial',
                    'fontSize':
                    '{}'.format
                    (self.game.canvas.w
                    idth / 70),
                    'left':
                    self.game.orthoX
                    (-7/16 *
                    orthoWidth), 'top':
                    self.game.orthoY
                    (fieldHeight // 2 +
                    self.hintShift)
            }))

            self.image = __new__
            (fabric.Line ([
                    self.game.orthoX (-
                    orthoWidth // 2),
                    self.game.orthoY
                    (fieldHeight // 2),
                    self.game.orthoX
                    (orthoWidth // 2),
                    self.game.orthoY
                    (fieldHeight // 2)
                ],
                {'stroke': 'white'}
            ))
```

```
        def increment (self,
        playerIndex):
            self.scores [playerIndex]
            += 1

        def reset (self):
            self.scores = [0, 0]
            Attribute.reset (self)  #
            Only does a commit here

        def commit (self):          #
        Committing labels is adapting
        their texts
            self.scoreLabels = [__new__
            (fabric.Text ('{}'.format
            (score), {
                    'fill': 'white',
                    'fontFamily':
                    'arial',
                    'fontSize':
                    '{}'.format
                    (self.game.canvas.w
                    idth / 30),
                    'left':
                    self.game.orthoX
                    (position *
                    orthoWidth), 'top':
                    self.game.orthoY
                    (fieldHeight // 2 +
                    self.nameShift)
            })) for score, position in
            zip (self.scores, (-2/16,
```

```
            6/16))]

        def draw (self):
            for playerLabel, scoreLabel
            in zip (self.playerLabels,
            self.scoreLabels):
                self.game.canvas.add
                (playerLabel)
                self.game.canvas.add
                (scoreLabel)
                self.game.canvas.add
                (self.hintLabel)
            self.game.canvas.add
            (self.image)

    class Game:
        def __init__ (self):
            self.serviceIndex = 1 if
            Math.random () > 0.5 else 0
               # Index of player that
            has initial service
            self.pause = True
                              # Start
            game in paused state
            self.keyCode = None

            self.textFrame =
            document.getElementById
            ('text_frame')
            self.canvasFrame =
            document.getElementById
            ('canvas_frame')
```

```
        self.buttonsFrame =
        document.getElementById
        ('buttons_frame')

        self.canvas = __new__
        (fabric.Canvas ('canvas',
        {'backgroundColor':
        'black', 'originX':
        'center', 'originY':
        'center'}))
        self.canvas.onWindowDraw =
        self.draw         # Install
        draw callback, will be
        called asynch
        self.canvas.lineWidth = 2
        self.canvas.clear ()

        self.attributes = []
                          # All
        attributes will insert
        themselves here
        self.paddles = [Paddle
        (self, index) for index in
        range (2)]     # Pass game
        as parameter self
        self.ball = Ball (self)
        self.scoreboard =
        Scoreboard (self)

        window.setInterval
        (self.update, 10)     #
        Install update callback,
        time in ms
```

```
        window.setInterval
        (self.draw, 20)       #
        Install draw callback, time
        in ms
        window.addEventListener
        ('keydown', self.keydown)
        window.addEventListener
        ('keyup', self.keyup)

        self.buttons = []

        for key in ('A', 'Z', 'K',
        'M', 'space', 'enter'):
            button =
            document.getElementById
            (key)
            button.addEventListener
            ('mousedown', (lambda
            aKey: lambda:
            self.mouseOrTouch
            (aKey, True)) (key))  #
            Returns inner lambda
            button.addEventListener
            ('touchstart', (lambda
            aKey: lambda:
            self.mouseOrTouch
            (aKey, True)) (key))
            button.addEventListener
            ('mouseup', (lambda
            aKey: lambda:
            self.mouseOrTouch
            (aKey, False)) (key))
```

```
                button.addEventListener
                ('touchend', (lambda
                aKey: lambda:
                self.mouseOrTouch
                (aKey, False)) (key))
                button.style.cursor =
                'pointer'
                button.style.userSelect
                = 'none'
                self.buttons.append
                (button)

            self.time = + __new__
            (Date)

            window.onresize =
            self.resize
            self.resize ()

        def install (self):
            for attribute in
            self.attributes:
                attribute.install ()

        def mouseOrTouch (self, key,
        down):
            if down:
                if key == 'space':
                    self.keyCode =
                    space
                elif key == 'enter':
                    self.keyCode =
                    enter
```

```
                else:
                    self.keyCode = ord
                    (key)
            else:
                self.keyCode = None

        def update (self):
                    # Note that update
        and draw are not synchronized
            oldTime = self.time
            self.time = + __new__
            (Date)
            self.deltaT = (self.time -
            oldTime) / 1000.

            if self.pause:
                        # If in paused
            state
                if self.keyCode ==
                space:              #   If
                spacebar hit
                    self.pause = False
                                #
                        Start playing
                elif self.keyCode ==
                enter:          #   Else
                if enter hit

                    self.scoreboard.res
                    et ()           #
                        Reset score
            else:
                        # Else, so if
```

```
            in active state
                for attribute in
                self.attributes:   #
                Compute predicted
                values
                    attribute.predict
                    ()

                for attribute in
                self.attributes:   #
                Correct values for
                bouncing and scoring
                    attribute.interact
                    ()

                for attribute in
                self.attributes:   #
                Commit them to pyglet
                for display
                    attribute.commit ()

        def scored (self, playerIndex):
                    # Player has scored
            self.scoreboard.increment
            (playerIndex) # Increment
            player's points
            self.serviceIndex = 1 -
            playerIndex     # Grant
            service to the unlucky
            player

            for paddle in self.paddles:
                        # Put paddles
```

```
            in rest position
                paddle.reset ()

            self.ball.reset ()
                            # Put ball in
            rest position
            self.pause = True
                           # Wait for next
            round

        def commit (self):
            for attribute in
            self.attributes:
                attribute.commit ()

        def draw (self):
            self.canvas.clear ()
            for attribute in
            self.attributes:
                attribute.draw ()

        def resize (self):
            self.pageWidth =
            window.innerWidth
            self.pageHeight =
            window.innerHeight

            self.textTop = 0

            if self.pageHeight > 1.2 *
            self.pageWidth:
                self.canvasWidth =
                self.pageWidth
```

```
                self.canvasTop =
                self.textTop + 300
            else:
                self.canvasWidth = 0.6
                * self.pageWidth
                self.canvasTop =
                self.textTop + 200

            self.canvasLeft = 0.5 *
            (self.pageWidth -
            self.canvasWidth)
            self.canvasHeight = 0.6 *
            self.canvasWidth

            self.buttonsTop =
            self.canvasTop +
            self.canvasHeight + 50
            self.buttonsWidth = 500

            self.textFrame.style.top =
            self.textTop;
            self.textFrame.style.left =
            self.canvasLeft + 0.05 *
            self.canvasWidth
            self.textFrame.style.width
            = 0.9 * self.canvasWidth

            self.canvasFrame.style.top
            = self.canvasTop
            self.canvasFrame.style.left
            = self.canvasLeft
            self.canvas.setDimensions
            ({'width':
```

```
            self.canvasWidth, 'height':
            self.canvasHeight})

            self.buttonsFrame.style.top
            = self.buttonsTop

            self.buttonsFrame.style.lef
            t = 0.5 * (self.pageWidth -
            self.buttonsWidth)

            self.buttonsFrame.style.wid
            th = self.canvasWidth

            self.install ()
            self.commit ()
            self.draw ()

        def scaleX (self, x):
            return x *
            (self.canvas.width /
            orthoWidth)

        def scaleY (self, y):
            return y *
            (self.canvas.height /
            orthoHeight)

        def orthoX (self, x):
            return self.scaleX (x +
            orthoWidth // 2)

        def orthoY (self, y):
```

```
            return self.scaleY
            (orthoHeight - fieldHeight
            // 2 - y)

        def keydown (self, event):
            self.keyCode =
            event.keyCode

        def keyup (self, event):
            self.keyCode = None

    game = Game ()  # Create and run
    game
```

Back to index

D3.js

D3.js is a graphics library that offers data driven animation of the DOM. Using Python's 'classical' object orientation, D3.js programs are easy to comprehend and maintain.

Run the 'd3js_demo' example

Code in d3js_demo/d3js_demo.html:

```
<html>
    <head>
        <style>
            body {background-image:
            url("water.jpg");}
            rect {fill: none;
            pointer-events: all;}
            .node {fill: #000000;}
            .cursor {fill: none;
            stroke: red; pointer-
            events: none;}
            .link {stroke:
            #003300;}
        </style>
    </head>
    <body style="font-family:arial;
    font-size:24; color:yellow">
        <div
        style="position:absolute;
        top:50; left:100">
```

```
                Click in frame to
                produce "frog spawn"...
                <br>
                Dots inside circle will
                be connected.
            </div>
            <script
            src="https://cdnjs.cloudfla
            re.com/ajax/libs/d3/3.4.1/d
            3.min.js"></script>
            <script
            type="module">import * as
            d3js_demo from
            './__target__/d3js_demo.js'
            ;</script>
        </body>
    </html>
```

Code in d3js_demo/d3js_demo.py:

```
    class Spawn:
        def __init__ (self, width,
        height):
            self.width, self.height,
            self.spacing = self.fill =
            width, height, 100,
            d3.scale.category20 ()
```

```
            self.svg = d3.select
            ('body'
            ) .append ('svg'
            ) .attr ('width',
            self.width
            ) .attr ('height',
            self.height
            ) .on ('mousemove',
            self.mousemove
            ) .on ('mousedown',
            self.mousedown)

            self.svg.append ('rect'
            ) .attr ('width',
            self.width
            ) .attr ('height',
            self.height)

            self.cursor =
            self.svg.append ('circle'
            ) .attr ('r', self.spacing
            ) .attr ('transform',
            'translate ({}, {})'
            .format (self.width / 2,
            self.height / 2)
            ) .attr ('class', 'cursor')

            self.force =
            d3.layout.force (
            ) .size ([self.width,
            self.height]
            ) .nodes ([{}]
```

```
            ) .linkDistance
            (self.spacing
            ) .charge (-1000
            ) .on ('tick', self.tick)


            self.nodes, self.links,
            self.node, self.link =
            self.force.nodes (),
            self.force.links (),
            self.svg.selectAll
            ('.node'),
            self.svg.selectAll
            ('.link')

            self.restart ()

        def mousemove (self):
            self.cursor.attr
            ('transform', 'translate ('
            + d3.mouse (self.svg.node
            ()) + ')')

        def mousedown (self):
            def pushLink (target):
                x, y = target.x -
                node.x, target.y -
                node.y
                if Math.sqrt (x * x + y
                * y) < self.spacing:
                    spawn.links.push
                    ({'source': node,
                    'target': target})
```

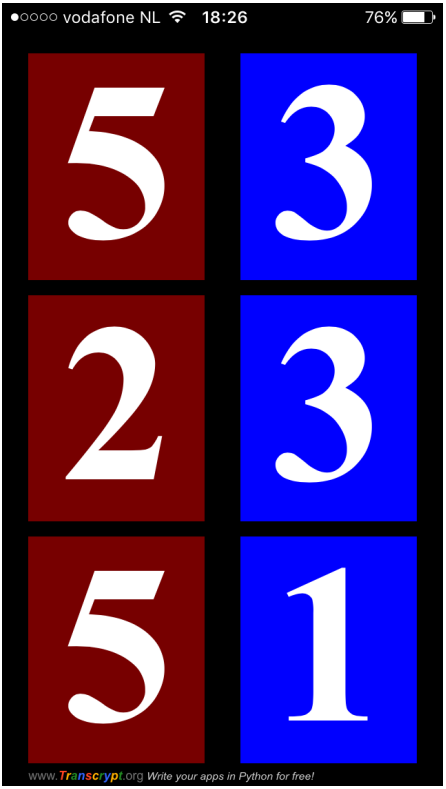
```
            point = d3.mouse
            (self.svg.node ())
            node = {'x': point [0],
            'y': point [1]}
            self.nodes.push (node)
            self.nodes.forEach
            (pushLink)
            self.restart ()

        def tick (self):
            self.link.attr ('x1',
            lambda d: d.source.x
            ) .attr ('y1', lambda d:
            d.source.y
            ) .attr ('x2', lambda d:
            d.target.x
            ) .attr ('y2', lambda d:
            d.target.y)

            self.node.attr ('cx',
            lambda d: d.x
            ) .attr ('cy', lambda d:
            d.y)

        def restart (self):
            self.link = self.link.data
            (self.links)

            self.link.enter (
            ) .insert ('line', '.node'
            ) .attr('class', 'link')
```

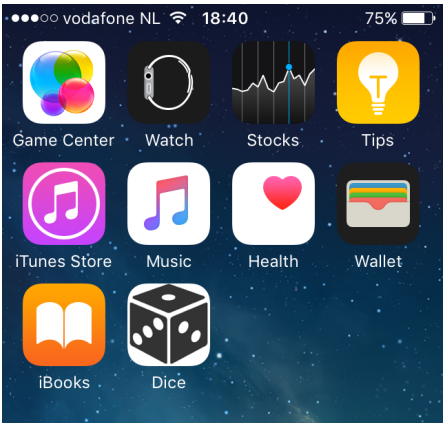
```
            self.node = self.node.data
            (self.nodes)

            self.node.enter (
            ) .insert ('circle',
            '.cursor'
            ) .attr ('class', 'node'
            ) .attr ('r', 7
            ) .call (self.force.drag)

            self.force.start ()

    spawn = Spawn (window.innerWidth,
    window.innerHeight)
```

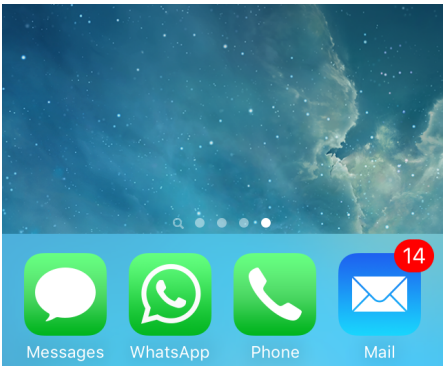
Back to index

'Dice' iOS Web App

Web Apps for iOS are full screen iOS browser applications that have a hidden address bar, while zooming and scrolling are blocked. By including a so called "cache manifest", they are cached on your iOS device, they don't need an internet connection to function. Also they can access the user's location. Starting them up happens by clicking a customized icon on your home screen. In other words: They behave like an app, but without the hassle. Rather than requiring admission to the app store, they can be downloaded from any site. With Transcrypt you just program them in Python. As an extra, they also feel at home in any browser, not just running under iOS.


●○○○○ vodafone NL 18:26 76%
5
3
2
3
5
1
www.Transcrypt.org Write your apps in Python for free!


●●●○○ vodafone NL 18:40 75%
Game Center
Watch
Stocks
Tips
iTunes Store
Music
Health
Wallet
iBooks
Dice



Run the 'ios_app' example

Code in ios_app/ios_app.html:

```
<html manifest="cache.manifest">
    <!-- v45 -->
    <head>
        <meta name="apple-mobile-
        web-app-capable"
        content="yes">
        <meta name="apple-mobile-
        web-app-status-bar-style"
        content="black">
        <meta name="apple-mobile-
        web-app-title"
        content="Dice">
        <meta name="viewport"
        content="width=device-
        width, initial-scale=1,
        user-scalable=no">
        <link rel="apple-touch-
        icon" href="ios_app.png">
```

```
        </head>
        <body>
            <script type="module">
                import * as ios_app
                from
                "./__target__/ios_app.j
                s"; window.ios_app =
                ios_app;
            </script>
        </body>
    </html>
```

Code in ios_app/ios_app.py:

```
    import random

    class Dice:
        def __init__ (self):

            document.body.addEventListe
            ner ('touchstart', lambda
            event: event.preventDefault
            ())

            document.body.addEventListe
            ner ('mousedown', lambda
            event: event.preventDefault
            ())
            document.body.style.margin
            = 0

            document.body.style.overflo
```

```
            w = 'hidden';

            self.all =
            document.createElement
            ('div')
            self.all.style.color =
            'white'

            self.all.style.backgroundCo
            lor = 'black'
            self.all.style.height =
            '100%'
            self.all.style.width =
            '100%'
            self.all.style.padding = 0
            self.all.style.margin = 0
            document.body.appendChild
            (self.all)

            self.dices = []

            for index in range (6):
                dice =
                document.createElement
                ('div')
                dice.normalColor =
                '#770000' if index < 3
                else '#0000ff'
                dice.style.position =
                'absolute'

                dice.style.backgroundCo
                lor = dice.normalColor
```

```
                dice.addEventListener
                ('touchstart', (lambda
                aDice: lambda:
                self.roll (aDice))
                (dice))  # Returns
                inner lambda
                dice.addEventListener
                ('mousedown', (lambda
                aDice: lambda:
                self.roll (aDice))
                (dice))
                self.dices.append
                (dice)
                self.all.appendChild
                (dice)

                dice.inner =
                document.createElement
                ('div')
                dice.inner.setAttribute
                ('unselectable', 'on')

                dice.inner.style.fontWe
                ight = 'bold'

                dice.inner.style.textAl
                ign = 'center'

                dice.inner.style.positi
                on = 'absolute'
                dice.inner.innerHTML =
                '?'
```

```
            dice.appendChild
            (dice.inner)

        self.banner =
        document.createElement
        ('div')
        self.banner.style.position
        = 'absolute'
        self.banner.style.cursor =
        'pointer'

        self.banner.addEventListene
        r ('touchstart',
        self.gotoTranscryptSite)

        self.banner.addEventListene
        r ('mousedown',
        self.gotoTranscryptSite)

        self.banner.style.fontFamil
        y = 'arial'
        self.banner.innerHTML = (
            '<span
            id="bannerLarge"><font
            color="777777">www.<b>
            <i>' +
            '<font
            color="ff4422">T<font
            color="ffb000">r<font
            color="228822">a<font
            color="3366ff">n' +
            '<font
            color="ff4422">s<font
```

```
                color="ffb000">c<font
                color="228822">r<font
                color="3366ff">y<font
                color="ffb000">p<font
                color="228822">t' +
                '</i></b><font
                color="777777">.org<fon
                t size={}><font
                color="cccccc"></span>'
                +
                '<span
                id="bannerSmall"><i>
                Write your apps in
                Python for free!</i>
                </span>'
            )
            self.all.appendChild
            (self.banner)

            self.bannerLarge =
            document.getElementById
            ('bannerLarge')
            self.bannerSmall =
            document.getElementById
            ('bannerSmall')

            self.audio = __new__ (Audio
            ('ios_app.mp3'))

            window.onresize =
            self.rightSize
            self.rightSize ()
```

```
        def gotoTranscryptSite (self):
            document.location.href =
            'http://www.transcrypt.org'

        def roll (self, dice):
            frameIndex = 10

            self.audio.play ()

            def frame ():
                nonlocal frameIndex
                frameIndex -= 1

                dice.inner.innerHTML =
                random.randint (1, 6)

                if frameIndex:
                    dice.style.color =
                    random.choice
                    (('red', 'green',
                    'blue', 'yellow'))
                    setTimeout (frame,
                    100)
                else:

                    dice.style.backgrou
                    ndColor =
                    dice.normalColor
                    dice.style.color =
                    'white'

            frame ()
```

```
        def rightSize (self):
            self.pageWidth =
            window.innerWidth
            self.pageHeight =
            window.innerHeight
            portrait = self.pageHeight
            > self.pageWidth

            for index, dice in
            enumerate (self.dices):
                if self.pageHeight >
                self.pageWidth:    #
                Portrait
                    dice.style.height =
                    0.3 *
                    self.pageHeight
                    dice.style.width =
                    0.4 *
                    self.pageWidth
                    dice.style.top =
                    (0.03 + (index if
                    index < 3 else
                    index - 3) * 0.32)
                    * self.pageHeight
                    dice.style.left =
                    (0.06 if index < 3
                    else 0.54) *
                    self.pageWidth

                    charBoxSide = 0.3 *
                    self.pageHeight

                    dice.inner.style.to
```

```
                    p = 0.15 *
                    self.pageHeight -
                    0.6 * charBoxSide

                    dice.inner.style.le
                    ft = 0.2 *
                    self.pageWidth -
                    0.5 * charBoxSide


                    self.banner.style.t
                    op = 0.975 *
                    self.pageHeight

                    self.banner.style.l
                    eft = 0.06 *
                    self.pageWidth


                    self.bannerLarge.st
                    yle.fontSize =
                    0.017 *
                    self.pageHeight

                    self.bannerSmall.st
                    yle.fontSize =
                    0.014 *
                    self.pageHeight
                else:
                                #
                Landscape
                    dice.style.height =
                    0.4 *
```

```
                    self.pageHeight
                    dice.style.width =
                    0.3 *
                    self.pageWidth
                    dice.style.top =
                    (0.06 if index < 3
                    else 0.54) *
                    self.pageHeight
                    dice.style.left =
                    (0.03 + (index if
                    index < 3 else
                    index - 3) * 0.32)
                    * self.pageWidth

                    charBoxSide = 0.4 *
                    self.pageHeight

                    dice.inner.style.to
                    p = 0.2 *
                    self.pageHeight -
                    0.6 * charBoxSide

                    dice.inner.style.le
                    ft = 0.15 *
                    self.pageWidth -
                    0.5 * charBoxSide


                    self.banner.style.t
                    op = 0.95 *
                    self.pageHeight

                    self.banner.style.l
```

```
                    eft = 0.03 *
                    self.pageWidth

                    self.bannerLarge.st
                    yle.fontSize =
                    0.015 *
                    self.pageWidth

                    self.bannerSmall.st
                    yle.fontSize =
                    0.012 *
                    self.pageWidth

                dice.inner.style.height
                = charBoxSide
                dice.inner.style.width
                = charBoxSide

                dice.inner.style.fontSi
                ze = charBoxSide

dice = Dice ()
```

Code in ios_app/cache.manifest:

```
```

```
CACHE MANIFEST

# v45

CACHE:
ios_app.html
ios_app.mp3
```

Back to index

Riot

The Riot library also uses a virtual DOM, just like React. It works with custom tags that define structure, style and behaviour of reusable components.

Run the 'riot_demo' example

Code in riot_demo/riot_demo.html:

```
<!DOCTYPE html>
<html>

  <head>
    <meta charset="UTF-8" />
    <title>Hello PyRiot ;-)</title>
    <!-- for transcrypted version
    we do not need the compiler.
         for the classic version
         you can use riot+compiler
         -->
```

```
    <script
    src="https://cdn.jsdelivr.net/r
    iot/2.5/riot.js"></script>
    <style>
        body {font-
        family:arial;font-
        size:30px;padding:50px;}
        h1 {font-
        size:50px;color:#0000ff;}
    </style>
</head>

<body>

<!-- classic riot (compiled on
server, could be done in browser)
        -->
        <sample id="s1"></sample>
        <script
        src="tags/sample.js">
        </script>

        <script>riot.mount('sample'
        )</script>


<!-- transcrypted version
(transpiled on server, using
python)            -->
        <sample2 id='s2'
        label="nr1"></sample2>
        <mp2     id='s3'
        label="nr2"></mp2>
```

```
            <!-- the transpiled tag's
            js -->
            <script type="module">
                import * as riot_demo
                from
                "./__target__/riot_demo
                .js"

                // register
                var cls =
                riot_demo.Sample2
                riot.tag2('sample2',
                cls.template,
                cls.style, '',
                function(opts) {
                    new cls(this,
                    opts)})
                document.t1 =
                riot.mount('sample2')
                [0].update()
                document.t2 =
                riot.mount('#s3',
                'sample2')[0]
                document.t2.update()
            </script>
    </body>
</html>
```

Code in riot_demo/color.py:

```
    # Producing a linear chain of style
    defs from a nested declaration of
    # color funcs
    msgs   = []
    styles = []

    debug = 0
    def _recurse(col, g, *s):
        msgs, styles, hsl = g
        lu = (('color', 0),
        ('background-color', 1))
        hsl = hsl[col]
        hsl = [hsl[:3], [hsl[0],
        hsl[1], hsl[3]]]
        css = ';'.join(
            [str(i) + ': hsl({}, {}%,
            {}%)'.format(*hsl[j]) for
            i, j in lu])
        for i in s:
            if debug:
                styles.append(col)
            else:
                styles.append(css)

            # empty string which we are
            replacing with value below,
            if not color
            # function follows
            msgs.append('%c')
            try:
                # can be another color
                function.. (triggering
                the recursion)
```

```
                i(g)
            except:
                # ... or normal output
                - in the current color
                msgs.pop() # TODO:
                msgs[-1] = ... instead
                pop() + append

                msgs.append('%c{}'.form
                at(i))

    # offering public, maybe of use.
    turned in _recurse into [fg, bg]
    # like red=[[0, 100, 90],[0, 100,
    50]]
    hsl = {'red'      : [  0, 100, 90,
    50],
           'orange'   : [ 39, 100, 85,
           50],
           'yellow'   : [ 60, 100, 35,
           50],
           'green'    : [120, 100, 60,
           25],
           'blue'     : [240, 100, 90,
           50],
           'purple'   : [300, 100, 85,
           25],
           'black'    : [  0,   0, 80,
           0],
           'gray'     : [237,   8, 80,
           50],
           }
```

```
    # generating the actual color
    functions, for each color:
    # right now we only now the color,
    later msgs and styles buffers:
    def _col(col): return lambda
    *parts: lambda g: _recurse(col, g,
    *parts)

    # TODO globals() not yet, so will
    import this in the clients:
    colors = {}
    # TODO .keys() currently necessary,
    should be easy to fix:
    for col in hsl.keys():
        colors[col] = _col(col)

    def cprint(*s):
        msgs, styles = [], []
        for i in s:
            i((msgs, styles, hsl))
        if debug:
            for i in range(len(msgs)):
                print (msgs[i], '-> ',
                styles[i])
        else:
            msg = ''.join(msgs)
            # FIXME this *crazy* eval
            is required since
            console.apply
            # is patched in Transcrypt
            st = '', ''.join(styles)
            st =
            ''.join(("console.log(\"",
```

```
            msg, '", "' + st + '")'))
            __pragma__('js', '{}',
            'eval(st)')

    # TODO if __name__ == '__main__'
    not works, could be cool for
    quicktests on the
    # server
    #     B, R, G = colors['blue'],
    colors['red'], colors['green']
    #     cprint(B('b1', 'b2', 'b3',
    R('r1', G('g2'), G('ga', 'gb'),
    'r2'), 'b2'))
```

Code in riot_demo/riot_tag.py:

```
    # Parent Class for a Transcrypt
    Riot Tag
    #
    # This binds the namespace of a
    riot tag at before-mount event 100%
    to that of
    # the a transcrypt instance, except
    members which begin with an
    underscore, those
    # are private to the transcrypt
    object.
```

```
#
# The 4 riot lifecycle events are
bound to overwritable python
functions.
#
# Immutables (strings, ints, ...)
are bound to property functions
within the tag,
# so the templates work, based on
state in the transcrypt tag.
# State can be changed in the riot
tag as well but take care to not
create new
# references - you won't find them
in the Transcrypt tag.
#
#
# Best Practices:
#  - mutate state only in the
transcrypt tag.
#  - declare all variables so they
are bound into the riot tag
#  - IF you declare new variables
to be used in templates, run
#    self.bind_vars(self.riot_tag)
# TODO: docstring format not
accepted by the transpiler,
strange.

__author__ = "Gunther Klessinger,
gk@axiros.com, Germany"
```

```
    # just a minihack to get some
    colors, mainly to test lamdas and
    imports:
    from color import colors, cprint as
    col_print
    c = colors
    M, I, L, R, B = c['purple'],
    c['orange'], c['gray'], c['red'],
    c['black']

    lifecycle_ev = ['before-mount',
    'mount', 'update', 'unmount']

    cur_tag_col = 0
    class RiotTag:
        """
        taking care for extending the
        riot tag obj with
        functions and immutable(!)
        properties of derivations of us
        See counter.
        """
        debug = None
        # placeholders:
        template = '<h1>it worx</h1>'
        style = ''
        node_name = 'unmounted'
        opts = None
        def __init__(self, tag, opts):
            # opts into the python
            instance, why not:
            self.opts = opts
            self._setup_tag(tag)
```

```
            # giving ourselves a unique
            color:
            global cur_tag_col #
            working (!)
            cur_tag_col = (cur_tag_col
            + 1) % len(colors)
            # TODO values() on a dict
            self.my_col =
            colors.items()[cur_tag_col]
            [1]

        def _setup_tag(self, tag):
            # keeping mutual refs
            tag.py_obj = self
            self.riot_tag = tag
            # making the event system
            call self's methods:
            handlers = {}
            for ev in lifecycle_ev:
                f = getattr(self,
                ev.replace('-', '_'))
                if f:
                    # this.on('mount',
                    function() {...}):
                    # whats nicer?
                    tag.on(ev, f)

        def pp(self, *msg):
            # color flash in the
            console. one color per tag
            instance.
            col_print(
```

```
                #B(self.riot_tag._riot_
                id),
                L('<',
                self.my_col(self.node_n
                ame, self.my_col), '/>
                '),
                M(' '.join([s for s in
                msg])))

        def _lifecycle_ev(self, mode):
            if self.debug:
                self.pp(mode + 'ing')

        # overwrite these for your
        specific one:
        def update (self):
        self._lifecycle_ev('update')
        def mount  (self):
        self._lifecycle_ev('mount')
        def unmount(self):
        self._lifecycle_ev('unmount')

        def before_mount(self):
            self._lifecycle_ev('before-
            mount')
            return self.bind_vars()

        def bind_vars(self):
            tag = self.riot_tag
            self.node_name =
            tag.root.nodeName.lower()
```

```
            self.debug and
            self.pp('binding vars')
            # binding self's functions
            into the tag instance
            # binding writable
            properties to everything
            else (e.g. ints, strs...)
            tag._immutables = im = []
            lc = lifecycle_ev
            for k in dir(self):
                # private or lifecycle
                function? don't bind:
                if k[0] == '_' or k in
                lifecycle_ev or k ==
                'before_mount':
                    continue
                v = getattr(self, k)
                # these I can't write
                in python. Lets use JS
                then.
                # TODO there should be,
                maybe some mocking
                facility for code
                # testing w/o a js
                runtime:
                __pragma__('js', '{}',
                '''
                        typeof v ===
                        "function" ||
                        typeof v ===
                        "object" ?
                        tag[k] = self[k]
                        :
```

```
                    tag._immutables.p
                    ush(k)''')

        __pragma__('js', '{}', '''
        var i = tag._immutables, py
        = self
        i.forEach(function(k, j, i)
        {

            Object.defineProperty(t
            ag, k, {
                get: function()  {
                return self[k]},
                set: function(v) {
                self[k] = v }
            })
        })''')
```

Code in riot_demo/riot_demo.py:

```
# an example user tag, using
RiotTag

from riot_tag import RiotTag

class P(RiotTag):
    debug = 1
    # never do mutables on class
    level. this is just to check if
    transpiler
```

```
        # creates the same behaviour -
        and it does, a second tag
        instance gets
        # the same lv object:
        lv = [{'name': 'n0'}]
        # immuatble on class level.
        does a second instance start at
        1?
        # answer: yes, perfect:
        counter = 1

        template = ''' <div><h1>Riot
        Transcrypt Tag Instance {label}
        </h1>

                                <div>INNER
                                </div>
                                </div> '''
        def count_up(self):
            self.counter = self.counter
            + 1
            self.pp('counter:',
            self.counter, 'len lv:',
            len(self.lv), 'adding one
            lv' )
            self.lv.append({'name': 'n'
            + self.counter})
            return self.counter

    # try some inheritance...
    class Sample2(P):
        # ... and change the state at
        every update, just for fun:
```

```
        template =
        P.template.replace('INNER', '''
        <div>
        <h5 each="{lv}">name: {name} -
        counter: {count_up()}</h5>
        </div>
        ''')

        # no scoped styles currently
        style = '''sample2 h5 {color:
        green}'''


        def __init__(self, tag, opts):
            self.label =
            opts.label.capitalize()  #
            this rocks so much.
            # alternative to super:
            RiotTag.__init__(self, tag,
            opts)
            # uncomment next line and
            chrome will stop:
            # debugger
            self.pp('tag init', 'adding
            2 lv')
            # mutating the lv object:
            self.lv.extend([{'name':
            'n1'}, {'name': 'n2'}])


        def update(self):
            self.pp('update handler in
            the custom tag, calling
```

```
                super')
                RiotTag.update(self)
```

Back to index

Plotly: live scientific plotting in the browser

The free version of the plotly.js library offers a rich set of scientific plots. As with any JavaScript library, Transcrypt is able to use plotly.js directly in the browser. There's no need to go through a special Python interface layer as would be the case when running plotly from a desktop Python installation. This means that the full native functionality of plotly.js is at your disposal, using plain Python syntax. For convenience, *__pragma__ ('jskeys')* can be used to void the need for quotes in dictionary keys.

It should be stressed that your complete computational application runs in the browser. While this restricts the scale of computations to what a browser running JavaScript can

handle, it offers the possibility to change computation parameters live and see the result immediately. Rather than putting the *outcomes* of your computations on the internet, you can make available the live computations themselves.

To achieve presentation quality results, plotly.js uses WebGL for its 3D graphs, something not all browsers currently support. The compiled Transcrypt code to use plotly.js is tiny and fast, but the plotly.js library itself has a large download and performs a lot of time consuming arithmetic. **So loading the example page below may take quite some time.**

Run the 'plotly_demo' example

Code in plotly_demo/plotly_demo.html:

```
<html>
    <head>
        <style>
            div {
                display: inline-
                block;
                overflow: hidden;
                padding: 10;
                width: 500;
```

```
                    height: 500;
                    border: 1px solid
                    gray;
                }
            </style>
            <script
            src="https://cdn.plot.ly/pl
            otly-latest.js"
            charset="UTF-8"></script>
        </head>
        <body>
            <div id="linear"></div>
            <div id="logarithmic">
            </div>
            <div id="polar"></div>
            <div id="wireframe"></div>
            <div id="ribbon"></div>
            <div id="surface"></div>
            <div id="bar"></div>
            <div id="pie"></div>
            <div id="scatter3d"></div>
            <script type="module">
                import * as hello from
                "./__target__/plotly_de
                mo.js";
            </script>
        </body>
    </html>
```

Code in plotly_demo/plotly_demo.py:

```
__pragma__ ('jskeys')   # For
convenience, allow JS style
unquoted string literals as
dictionary keys

import random
import math
import itertools

xValues = [2 * math.pi * step / 200
for step in range (201)]
yValuesList = [
    [math.sin (xValue) + 0.5 *
    math.sin (xValue * 3 + 0.25 *
    math.sin (xValue * 5)) for
    xValue in xValues],
    [1 if xValue <= math.pi else -1
    for xValue in xValues]
]
kind = 'linear'
Plotly.plot (
    kind,
    [
        {
            x: xValues,
            y: yValues
        }
        for yValues in yValuesList
    ],
    {
        title: kind,
        xaxis: {title: 'U (t)
        [V]'},
```

```
            yaxis: {title: 't [s]'}
        }
    )

    try:
        xValues = list (range (10))
        yValues = [math.exp (x**2) for
        x in xValues]
        kind = 'logarithmic'
        Plotly.plot (
            kind,
            [
                {
                    x: xValues,
                    y: yValues
                }
            ],
            {
                title: kind,
                xaxis: {title: 'x'},
                yaxis: {type: 'log',
                tickformat: '2e',
                title: 'exp (x**2)'}
            }
        )
    except: # Microsoft Edge bug in exp
    function
        pass

    tangentialValues = list (range
    (-180, 180))
    radialValuesList = [
```

```
        [abs (t) for t in
        tangentialValues],
        [180 - abs (t) for t in
        tangentialValues],
        [abs (2 * t) for t in
        tangentialValues]
    ]
    kind = 'polar'
    Plotly.plot (
        kind,
        [
            {
                t: tangentialValues,
                r: radialValues,
                name: 'Cardioid
                {}'.format (i),
            }
            for i, radialValues in
            enumerate
            (radialValuesList)
        ],
        {
            title: kind
        }
    )

    denseGrid = [8 * math.pi * step /
    200 for step in range (-100, 101)]
    sparseGrid = [8 * math.pi * step /
    200 for step in range (-100, 101,
    10)]

    def getZValues (xGrid, yGrid):
```

```
        return [
            [math.sin (r) / r for  r in
            [math.sqrt (x * x + y * y)
            for x in xGrid]] # One row
            for y in yGrid

                                # For all
            rows
        ]

    kind = 'wireframe'
    document.getElementById (kind)
    .innerHTML = 'Plotly {} not yet
    functional for JS6'.format (kind)

    aType = 'scatter3d'
    Plotly.plot (
        kind,
        itertools.chain (
            [
                {
                    x: denseGrid,
                    y: [sparseGrid [i]
                    for value in
                    denseGrid],
                    z: getZValues
                    (denseGrid,
                    sparseGrid) [i],
                    type: aType,
                    mode: 'lines',
                    line:
                    {color:'rgb(0,0,255
                    )'},
```

```
                    zmin: -0.2,
                    zmax: 1,
                    showscale: not i,
                }
                for i in range (20)
            ],
            [
                {
                    x: [sparseGrid [i]
                    for value in
                    denseGrid],
                    y: denseGrid,
                    z: zip (*getZValues
                    (sparseGrid,
                    denseGrid)) [i],
                    # Poor man's
                    transpose to avoid
                    dependency of demo
                    on Numscrypt
                    type: aType,
                    mode: 'lines',
                    line:
                    {color:'rgb(0,0,255
                    )'},
                    zmin: -0.2,
                    zmax: 1,
                    showscale: not i,
                } for i in range (20)
            ]
        ),
        {
            title: kind,
            showlegend: False
```

```
        }
    )

    kind = 'ribbon'
    Plotly.plot (
        kind,
        [
            {
                x: denseGrid,
                y: list (range (i * 20,
                (i + 0.7) * 20)),
                z: getZValues
                (denseGrid, denseGrid)
                [i * 20 : (i + 0.7) *
                20], # Take the right
                'band' out of the data
                type: 'surface',
                zmin: -0.2,
                zmax: 1,
                showscale: not i,
            }
            for i in range (10)
        ],
        {
            title: kind
        }
    )

    kind = 'surface'
    Plotly.plot (
        kind,
        [
            {
```

```
                x: denseGrid,
                y: denseGrid,
                z: getZValues
                (denseGrid, denseGrid),
                type: kind,
                zmin: -0.2,
                zmax: 1
            }
        ],
        {
            title: kind
        }
    )

    labels = ['much', 'more', 'most']
    kind = 'bar'
    Plotly.plot (
        kind,
        [
            {
                name: 'rare',
                x: labels,
                y: [1, 2, 4],
                type: kind
            },
            {
                name: 'common',
                x: labels,
                y: [8, 16, 32],
                type: kind
            }
        ],
        {
```

```
            title: kind,
            barmode: 'group'
        }
    )

    kind = 'pie'
    Plotly.plot (
        kind,
        [
            {
                values: [1, 2, 3, 4, 5,
                6],
                labels: ['least',
                'less', 'little',
                'much', 'more',
                'most'],
                type: kind
            }
        ],
        {
            title: kind
        }
    )

    kind = 'scatter3d'
    def getRandoms (aMax):
        return [random.randint (0,
        aMax) for i in range (20)]
    Plotly.plot (
        kind,
        [
            {
                x: getRandoms (aMax),
```

```
                y: getRandoms (aMax),
                z: getRandoms (aMax),
                mode: 'markers',
                marker: {
                    color: 'rgb({},
                    127, {})'.format
                    (127 - aMax * 12,
                    aMax * 12),
                    size: 12,
                    symbol: 'circle',
                    line: {
                        color: 'rgb({},
                        255,
                        {})'.format
                        (255 - aMax *
                        25, aMax * 25),
                        width: 1
                    }
                },
                type: kind
            }
            for aMax in (2, 5, 10)
        ],
        {
            title: kind
        }
    )
```