Exercise Sheet 1

Exercise 1 - Wallis formula

Compute the decimals of Pi using the Wallis formula:

$$\pi = 2 \prod_{i=1}^{\infty} \frac{4i^2}{4i^2 - 1}$$

```python
pi = 3.14159265358979312

my_pi = 1.

for i in range(1, 100000):
    my_pi *= 4 * i ** 2 / (4 * i ** 2 - 1.)

my_pi *= 2

print(pi)
print(my_pi)
print(abs(pi - my_pi))
```

Exercise 2 - Fibonacci sequence

Write a function that displays the n first terms of the Fibonacci sequence, defined by:

$$\begin{cases} U_0 = 0 \\ U_1 = 1 \\ U_{n+2} = U_{n+1} + U_n \end{cases}$$

```python
def fib(n):
    """Display the n first terms of Fibonacci sequence"""
    a, b = 0, 1
    i = 0
    while i < n:
        print(b)
        a, b = b, a+b
        i +=1
```

Exercise 3 - Quicksort

Implement the quicksort algorithm, as defined by wikipedia

```
function quicksort(array)
    var list less, greater
    if length(array) < 2
        return array
    select and remove a pivot value pivot from array
    for each x in array
        if x < pivot + 1 then append x to less
        else append x to greater
    return concatenate(quicksort(less), pivot, quicksort(greater))
```

```python
"""
Implement the quick sort algorithm.
"""

def qsort(lst):
    """ Quick sort: returns a sorted copy of the list.
    """
    if len(lst) <= 1:
        return lst
    pivot, rest    = lst[0], lst[1:]

    # Could use list comprehension:
    # less_than      = [ lt for lt in rest if lt < pivot ]

    less_than = []
    for lt in rest:
        if lt < pivot:
            less_than.append(lt)

    # Could use list comprehension:
    # greater_equal  = [ ge for ge in rest if ge >= pivot ]

    greater_equal = []
    for ge in rest:
```

```
        if ge >= pivot:
            greater_equal.append(ge)
    return qsort(less_than) + [pivot] + qsort(greater_equal)

# And now check that qsort does sort:
assert qsort(range(10)) == range(10)
assert qsort(range(10)[::-1]) == range(10)
assert qsort([1, 4, 2, 5, 3]) == sorted([1, 4, 2, 5, 3])
```

Exercise 4 - Turtle graphics

This exercise uses the turtle module, which allows you to create images using turtle graphics. See the documentation for more details:

- https://docs.python.org/3.3/library/turtle.html?highlight=turtle

For example, here is how you would draw a square using turtle:

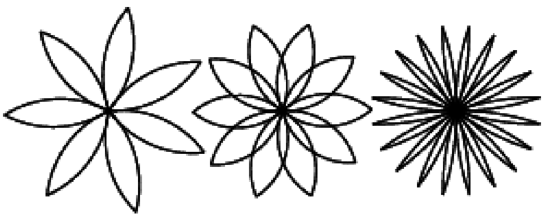
```
def square(t, length):
    for i in range(4):
        t.fd(length)
        t.lt(90)

square(bob, 100)
```

a) Make a copy of square and change the name to polygon. Add another parameter named n and modify the body so it draws an n-sided regular polygon. Hint: The exterior angles of an n-sided regular polygon are 360/n degrees.

b) Write a function called circle that takes a turtle, t, and radius, r, as parameters and that draws an approximate circle by calling polygon with an appropriate length and number of sides. Test your function with a range of values of r.

c) Make a more general version of circle called arc that takes an additional parameter angle, which determines what fraction of a circle to draw. angle is in units of degrees, so when angle=360, arc should draw a complete circle.



d) Write an appropriately general set of functions that can draw flowers as above.

```
"""This module contains a code example related to

Think Python, 2nd Edition
by Allen Downey
http://thinkpython2.com

Copyright 2015 Allen Downey

License: http://creativecommons.org/licenses/by/4.0/
"""

from __future__ import print_function, division

import math
import turtle

def polyline(t, n, length, angle):
    """Draws n line segments.

    t: Turtle object
    n: number of line segments
    length: length of each segment
    angle: degrees between segments
    """
    for i in range(n):
        t.fd(length)
        t.lt(angle)
```

```python
def polygon(t, n, length):
    """Draws a polygon with n sides.

    t: Turtle
    n: number of sides
    length: length of each side.
    """
    angle = 360.0/n
    polyline(t, n, length, angle)


def arc(t, r, angle):
    """Draws an arc with the given radius and angle.

    t: Turtle
    r: radius
    angle: angle subtended by the arc, in degrees
    """
    arc_length = 2 * math.pi * r * abs(angle) / 360
    n = int(arc_length / 4) + 3
    step_length = arc_length / n
    step_angle = float(angle) / n

    # making a slight left turn before starting reduces
    # the error caused by the linear approximation of the arc
    t.lt(step_angle/2)
    polyline(t, n, step_length, step_angle)
    t.rt(step_angle/2)


def circle(t, r):
    """Draws a circle with the given radius.

    t: Turtle
    r: radius
    """
    arc(t, r, 360)

def petal(t, r, angle):
    """Draws a petal using two arcs.

    t: Turtle
    r: radius of the arcs
    angle: angle (degrees) that subtends the arcs
    """
    for i in range(2):
        arc(t, r, angle)
        t.lt(180-angle)


def flower(t, n, r, angle):
    """Draws a flower with n petals.

    t: Turtle
    n: number of petals
    r: radius of the arcs
    angle: angle (degrees) that subtends the arcs
    """
    for i in range(n):
        petal(t, r, angle)
        t.lt(360.0/n)


def move(t, length):
    """Move Turtle (t) forward (length) units without leaving a trail.
    Leaves the pen down.
    """
    t.pu()
    t.fd(length)
    t.pd()

# the following condition checks whether we are
# running as a script, in which case run the test code,
# or being imported, in which case don't.

if __name__ == '__main__':
    bob = turtle.Turtle()

    # draw a circle centered on the origin
    radius = 100
    bob.pu()
    bob.fd(radius)
    bob.lt(90)
    bob.pd()
    circle(bob, radius)

    # wait for the user to close the window
    turtle.mainloop()
```