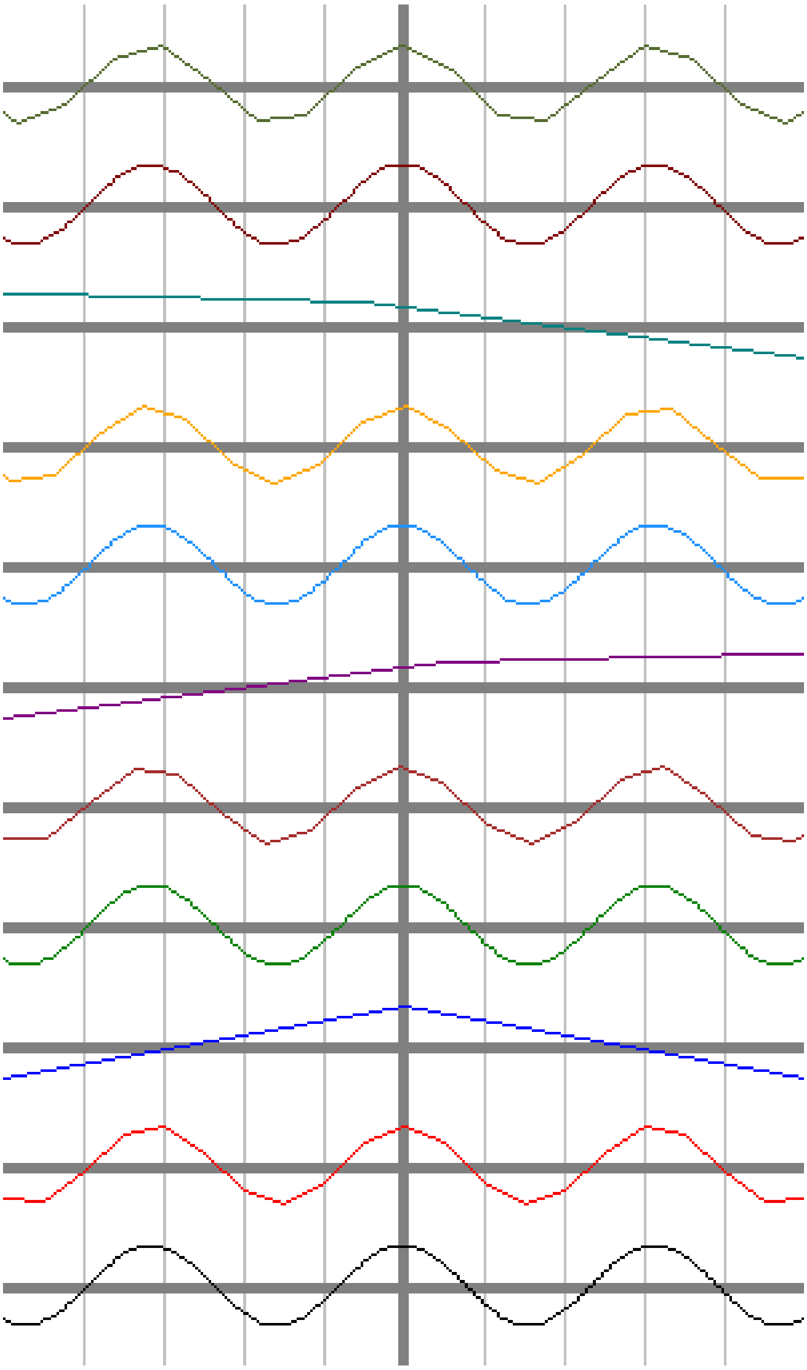
Contents

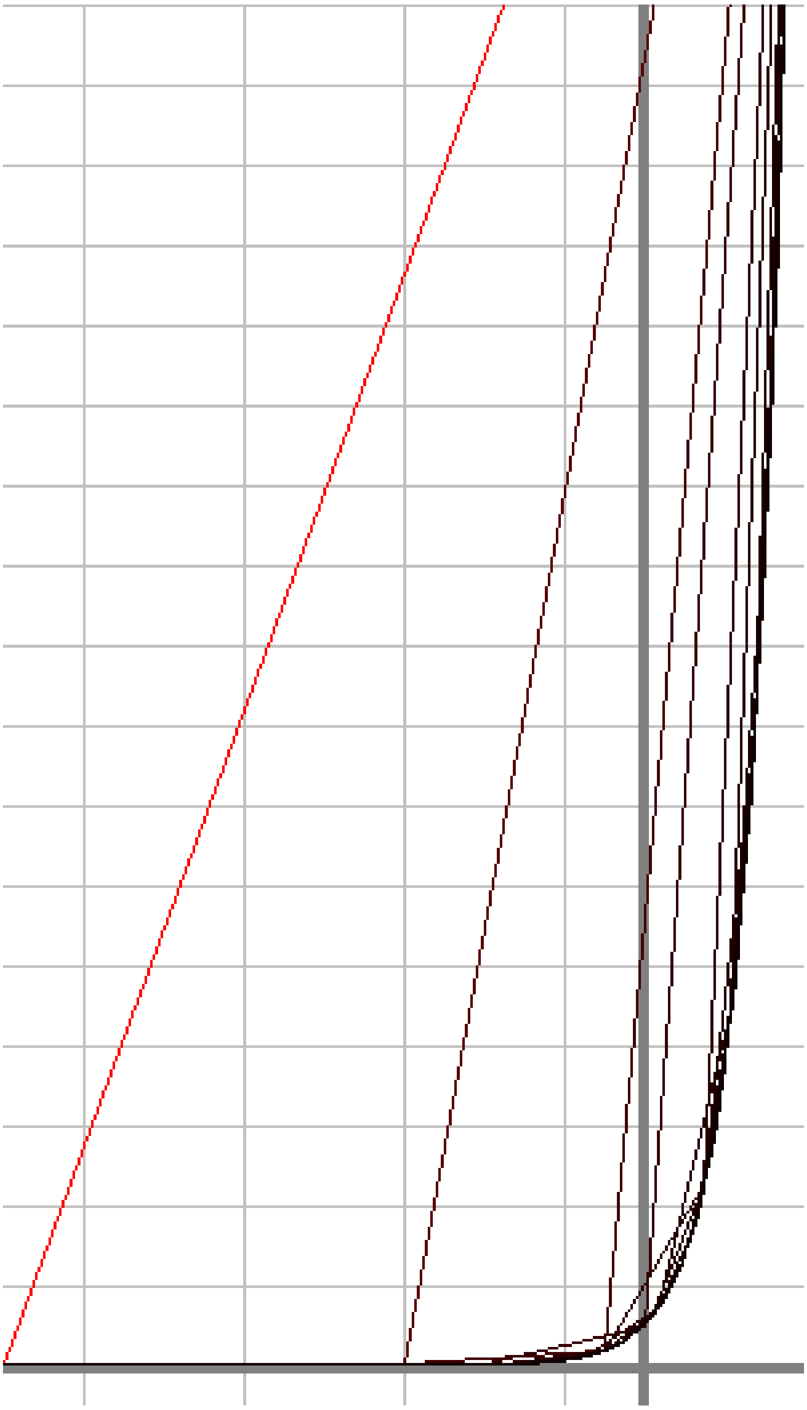
1 Basic Test Results

```
Starting tests...
Mon Jan  4 17:31:49 IST 2016
d738a083208bbd7e8d9b6fcef99db5d2e835640c  -


Archive:  /tmp/bodek.m4tG4o/intro2cs/ex11/elinorperl/presubmission/submission
  inflating: src/ex11.py
  inflating: src/README

Testing README...
Done testing README...

Running presubmit tests...
144 passed tests out of 144
result_code    div    144    1
144 passed tests out of 144
result_code    compose    144    1
15 passed tests out of 15
result_code    inverse    15    1
12 passed tests out of 12
result_code    integral    12    1
2 passed tests out of 2
result_code    single    2    1
11 passed tests out of 11
result_code    const    11    1
144 passed tests out of 144
result_code    mul    144    1
144 passed tests out of 144
result_code    sum    144    1
144 passed tests out of 144
result_code    sub    144    1
36 passed tests out of 36
result_code    deriv    36    1
Done running presubmit tests

Creating graph images:
Done...

Tests completed

Additional notes:

There will be additional tests which will not be published in advance.
```

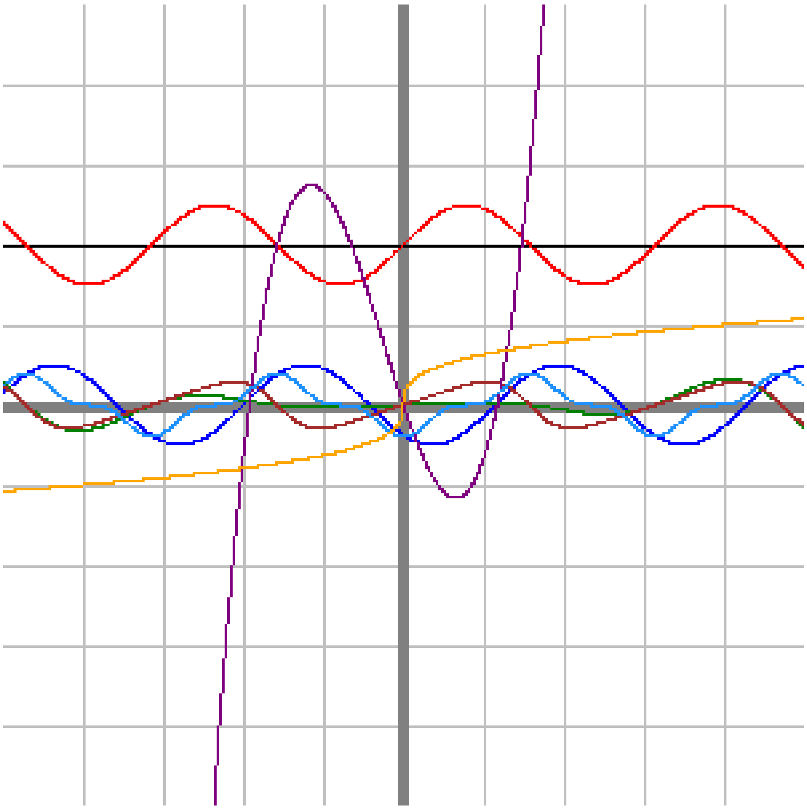
2 cos.png



3 exp.png



4 flist.png



5 README

```
elinorperl
329577464
Elinor Perl

I discussed the exercise with Eli Corn, Nophar Sarel, Talya Adams

=========================================
=  README for ex11: Math Functions  =
=========================================


==================
=  Description:  =
==================

In this exercise, I defined and mapped out different types of functions.
Thereafter graphing more complicated function using the basic functions
that I implemented.

======================
=  Special Comments  =
======================

I used stackoverflow.com
```

6 ex11.py

```python
#!/usr/bin/env python3

import math

EPSILON = 1e-5
DELTA = 1e-3
SEGMENTS = 100


def plot_func(graph, f, x0, x1, num_of_segments=SEGMENTS, c=’black’):
    """
    plot f between x0 and x1 using num_of_segments straight lines.
    use the plot_line function in the graph object.
    f will be plotted to the screen with color c.
    """
    segment_value = (x1 - x0)/num_of_segments
    while x0 < x1:
        graph.plot_line((x0,f(x0)),
                        (x0+segment_value, f(x0+segment_value)), c)
        x0 += segment_value


def const_function(c):
    """return the mathematical function f such that f(x) = c
    >>> const_function(2)(2)
    2
    >>> const_function(4)(2)
    4
    """
    return lambda x: c

def identity():
    """return the mathematical function f such that f(x) = x
    >>>identity()(3)
    3
    """
    return lambda x: x


def sin_function():
    """return the mathematical function f such that f(x) = sin(x)
    >>> sinF()(math.pi/2)
    1.0
    """
    return lambda x: math.sin(x)


def sum_functions(g, h):
    """return f s.t. f(x) = g(x)+h(x)"""
    return lambda x: g(x) + h(x)


def sub_functions(g, h):
    """return f s.t. f(x) = g(x)-h(x)"""
    return lambda x: g(x) - h(x)


def mul_functions(g, h):
    """return f s.t. f(x) = g(x)*h(x)"""
    return lambda x: g(x)*h(x)


def div_functions(g, h):
    return lambda x: g(x)/h(x)
    # The function solve assumes that f is continuous.
    # solve return None in case of no solution

def solve(f, x0=-10000, x1=10000, epsilon=EPSILON):
    """return the solution to f in the range between x0 and x1"""
    answer = True
    while answer:
        mid_point = (x1 + x0)/2
        if abs(f(mid_point)) < epsilon:
            return mid_point
        if f(x1)*f(mid_point) < 0:
            x0 = mid_point
        elif f(x0)*f(mid_point) < 0:
            x1 = mid_point
        else:
            return None

    # inverse assumes that g is continuous and monotonic.
def inverse(g, epsilon=EPSILON):
    """return f s.t. f(g(x)) = x"""
    x0 = -2
    x1 = 2
    inverse_func = lambda x: solve(sub_functions(g, const_function(x)),
                           epsilon=epsilon)
    while inverse_func is None:
        x1 **= 2
        x0 = -x1
        inverse_func = lambda x: solve\
            (sub_functions(g, const_function(x)), x0, x1, epsilon)
    return inverse_func


def compose(g, h):
    """return the f which is the compose of g and h """
    return lambda x: g(h(x))


def derivative(g, delta=DELTA):
    """return f s.t. f(x) = g'(x)"""
    return lambda x: (g(x+delta) - g(x))/delta


def definite_integral(f, x0, x1, num_of_segments=SEGMENTS):
    """
    return a float - the definite_integral of f between x0 and x1
    >>>definite_integral(const_function(3),-2,3)
    15
    """
    if x0 == 0 and x1 == 0:
        return 0
    segment_value = (x1 - x0)/num_of_segments
    sigma = 0
    xi = x0 + segment_value
    for _ in range(num_of_segments):
        sigma += f((x0+xi)/2)*(xi-x0)
        x0 += segment_value
        xi += segment_value
    return sigma

def integral_function(f, delta=0.01):
    """return F such that F'(x) = f(x)"""
    return lambda x: definite_integral(f, 0, x, int(math.ceil(abs(x)/delta))) \
        if x > 0 else -definite_integral(f, x, 0, int(math.ceil(abs(x)/delta)))
```

```
def ex11_func_list():
    """return a list of functions as a solution to q.12"""
    func_list = []
    id = identity()
    sin = sin_function()
    x_squared = mul_functions(id, id)
    cos = derivative(sin)
    # A constant function of 4, f(x) = 4
    func0 = const_function(4)
    # The sum of sin(x) and the constant 4, f(x) = sin(x) + 4
    func1 = sum_functions(sin, func0)
    # The composition of sin x and 4, f(x) = sin(x + 4)
    func2 = compose(sin, sum_functions(id, func0))
    # Multiplying sin and cos while dividing the cos by the constant 100
    # f(x) = sin(x)*(cos(x)/100)
    func3 = mul_functions(sin, div_functions(x_squared, const_function(100)))
    # Division of sin by cos and 2, f(x) = sin(x)/(cos(x)+2
    func4 = div_functions(sin, sum_functions(cos, const_function(2)))
    # The integral for f(x) = x**2 + x -3
    func5 = integral_function(sub_functions((sum_functions
                                             (mul_functions(id, id), id)),
                                            const_function(3)))
    # Multiplying and subtraction for f(x) = 5 * (sin(cos(x)) - cos(x))
    func6 = mul_functions(const_function(5), sub_functions(compose(sin, cos),
                                                           cos))
    # The inverse function for f(x) = x**3
    funct7 = inverse(mul_functions(x_squared, id))
    func_list = [func0, func1, func2, func3, func4, func5, func6, funct7]
    return func_list




# function that genrate the figure in the ex description
def example_func(x):
     return x**3


if __name__ == "__main__":
    import tkinter as tk
    from ex11helper import Graph
    master = tk.Tk()
    graph = Graph(master, -10, -10, 10, 10)
    # un-tag the line below after implementation of plot_func
    plot_func(graph, inverse(example_func),-10,10,SEGMENTS, ’blue’)
    plot_func(graph, example_func,-10,10,SEGMENTS, ’purple’)
    color_arr = [’black’, ’blue’, ’red’, ’green’, ’brown’, ’purple’,
                 ’dodger blue’, ’orange’]
    # un-tag the lines below after implementation of ex11_func_list
    # for f in ex11_func_list():
    #      plot_func(graph, f, -10, 10, SEGMENTS, ’red’)

    master.mainloop()
```