UNIVERSITY OF WATERLOO

Lesson 1.5: The Math Module
Introduction to Modules

As discussed in the previous lesson, we have created our own testing package called the `check` module. Recall that a module is a collection of functions that can be used. Our `check` module included functions like `expect` and `within` amongst others we will see later in this course.

It turns out that Python also has built-in modules as well. These are not automatically included partially for speed and to limit the bloat in Python. While the `check` module is already included in every block of code in this course (and as mentioned can be downloaded for use offline), other modules need to explicitly be included. This can be done using the command:

```
import module_name
```

For this course, you can `import check` without harm and will need to place this piece of code at the top of your coding blocks and files if working with the `check` module. In this lesson, we discuss another module, namely the `math` module. To use the math module, you will need to at the beginning of the file you are creating (or problem you are solving) include the line

```
import math
```

This will give you access to everything inside the `math` module. Most of the functions are self explanatory. We include below a short list of functions we will be interested in using in this course. Don't worry about memorizing the list! A reference sheet similar to the following will be given on examinations if needed.

Also below, there are some function which take in optional parameters which are denoted using square brackets. As an example, the function `math.log` takes in an optional `base` parameter which if left out defaults to e. As an explicit example, `math.log(10)` is equal to $\log_e(10)$ and `math.log(10, 3)` is equal to $\log_3(10)$. Parameters not contained in square brackets are not optional.

A good rule of thumb however is that the `math` module contains most 'advanced' arithmetic operations including the following:

List of Functions in Python Math Module

Function	Description
`acos(x)`	Returns the arc cosine of `x`.
`acosh(x)`	Returns the inverse hyperbolic cosine of `x`.
`asin(x)`	Returns the arc sine of `x`.
`asinh(x)`	Returns the inverse hyperbolic sine of `x`.

`atan(x)`	Returns the arc tangent of `x`.
`atan2(y, x)`	Returns the arc tangent of `y / x`.
`atanh(x)`	Returns the inverse hyperbolic tangent of `x`.
`ceil(x)`	Returns the smallest integer greater than or equal to `x`.
`cos(x)`	Returns the cosine of `x`.
`cosh(x)`	Returns the hyperbolic cosine of `x`.
`degrees(x)`	Converts angle `x` from radians to degrees.
`e`	Mathematical constant e (2.71828...).
`exp(x)`	Returns `e**x`.
`factorial(x)`	Returns the factorial of `x`.
`floor(x)`	Returns the largest integer less than or equal to `x`.
`gamma(x)`	Returns the Gamma function at `x`.
`log(x[, base])`	Returns the logarithm of `x` to the optional parameter base (default is base e).
`log2(x)`	Returns the base-2 logarithm of `x`.
`log10(x)`	Returns the base-10 logarithm of `x`.
`pi`	Mathematical constant, the ratio of circumference of a circle to it's diameter (3.14159...).
`pow(x, y)`	Returns `x` raised to the power `y`.
`radians(x)`	Converts angle `x` from degrees to radians.
`sin(x)`	Returns the sine of `x`.
`sinh(x)`	Returns the hyperbolic sine of `x`.
`sqrt(x)`	Returns the square root of `x`.
`tan(x)`	Returns the tangent of `x`.
`tanh(x)`	Returns the hyperbolic tangent of `x`.

We recommend using the above table if you want to find a function in the `math` module. If a mathematical function is needed in this course, it can be found above. To use a function, simply call `math.fn_name(params)`. For example, calling `math.cos(0)` will return the value `1.0`. There are two

constants, namely `math.e` and `math.pi` that can be used exactly as typed here. For example, we can set `tau = 2 * math.pi` and then use the variable `tau` later in our code.

A common error for students is dropping the `math` from a function. So for example, you cannot call `cos(0)` since `cos` is not a built in function — it lives in the math module. To call this correctly, you must `import math` at the top of your program and then call `math.cos(0)` which returns the value `1.0`.

Try out the following code in the visualizer before trying out some problems below!

```
1  import math
2
3
4  a = math.sqrt(25)
5  b = math.log(32,2)
6  c = math.log(32.0, 10)
7  d = math.floor(math.log(32.0, math.e))
8  e = math.factorial(10)
9  f = math.cos(math.pi)
```

Visualize Reset Code

Here's an example using the `math` module with our design recipe elements. Notice below is the first time we see the `pass` statement. This command does not do anything. It is what is known as a "no operation" or a "NO-OP" for short. It is included so that if you submit code immediately, Python thinks that the function is defined. A function without a body is not a proper function in Python and without some line(s) of code, Python will throw an error when submitted. This is meant to help prevent this error. We recommend removing it when you start but you can just as easily leave it in your code as well without harm. This statement isn't important for this course but since students often ask about it, we felt that a brief statement here explaining it would be beneficial.

Concept Check 1.5.1: Pythagorean Theorem

In the following sequence of problems, we will recall some geometry facts and write Python programs to compute some values. You won't need to memorize these for examinations but you might need these for homework problems!

Recall the Pythagorean Theorem, which states that for a right angle triangle with side lengths x, y and hypotenuse z, we have the following:

$$z^2 = x^2 + y^2$$

Write a function `third_side(opp, hyp)` which consumes two floating point side lengths in `opp` and `hyp`, the latter of which is the hypotenuse, and returns the length of the third side.

0/3 points
Attempts: 0 / Unlimited

```
def third_side(opp, hyp):
  '''
  Pythagorean Theorem given the hypotenuse
  in hyp and opposite side in opp

  third_side: Float Float -> Float
  Requires:
     0.0 < opp < hyp

  Examples:
     third_side(4.0, 5.0) => 3.0
  '''
  ##YOUR CODE GOES HERE
  pass
```

Concept Check 1.5.2: Sine

Given a right angle triangle, we can also apply trigonometry! Recall that for a right angle triangle, we have the following relationship

$$\sin(\theta) = \frac{\text{opp}}{\text{hyp}}$$

Write a function `hypotenuse(opp, theta)` which consumes the `opp` side length and the angle `theta` in radians and returns the length of the hypotenuse.

0/3 points
Attempts: 0 / Unlimited

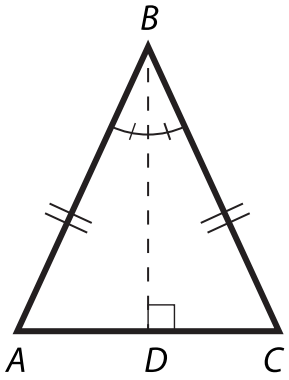
```
def hypotenuse(opp, theta):
  '''
  Returns the hypotenuse of a right angle triangle
  given an angle theta in radians and
  the opposite side length in opp.

  hypotenuse: Float Float -> Float
  Requires:
     0.0 < opp
     0.0 < theta < math.pi/2

  Examples:
     hypotenuse(1.0, math.pi/4) => 1.414213562373095
     hypotenuse(2.0, math.pi/6) => 4.0
  '''
  ##YOUR CODE GOES HERE
  pass
```

Concept Check 1.5.3: Isosceles Triangles

In an isosceles triangle, If we draw the altitude (the height) starting from the vertex contained between the two equal sides, it turns out that this line is not just the altitude but is also an angle bisector and the perpendicular bisector!



Write a function `height(doubled_side, base)` that consumes the side length of the repeated side and the length of the other side in `base` and returns the height of the triangle with respect to the `base` side length.

0/3 points
Attempts: 0 / Unlimited

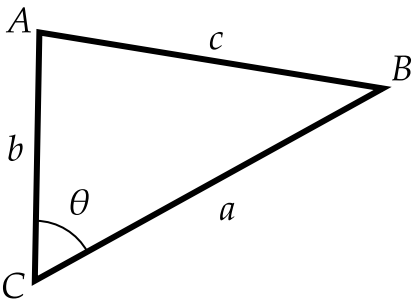
```
def height(doubled_side, base):
    '''
    Returns the height of the isosceles triangle
    with sides doubled_side, doubled_side and base

    height: Float Float -> Float
    Requires:
       0.0 < base/2 < doubled_side

    Example:
       height(5.0, 6.0) => 4.0
    '''
    ##YOUR CODE GOES HERE
    pass
```

Concept Check 1.5.4: Cosine Law

The cosine law gives us a mathematical way to compute the third side length of a triangle given two side lengths and the contained angle:



The formula is given by:

$$c^2 = a^2 + b^2 - 2ab\cos(\theta)$$

Write a function `cosine_law(a, b, theta)` that consumes two side lengths of a triangle and the contained angle `theta` and returns the length of the third side.

0/3 points
Attempts: 0 / Unlimited

```
def cosine_law(a, b, theta):
    '''
    Returns the third side length using
    the cosine law with side lengths a and b
    and contained angle theta in radians

    cosine_law: Float Float Float -> Float
    Requires:
       0.0 < a
       0.0 < b
       0.0 < theta < math.pi

    Example:
       cosine_law(3, 4, math.pi/3) => 3.60555127546
    '''
    ##YOUR CODE GOES HERE
    pass
```

Concept Check 1.5.5: Debugging

1 point possible (graded)
Consider the following code:

```
import math



def f(x):
    '''
    Returns 2*sqrt of (x+1)

    f: Float -> Float
    Requires: x > 0.0
    '''
    y = math.sqrt(x+1)
        z = 2*y
    return z
```

Which of the following are true? **Check all that apply.**

- [ ] There is an indentation error in `f`.

- [ ] You cannot use the value of `y` to initialize `z`.

Concept Check 1.5.6: Distance

1 point possible (graded)
Consider the following code which attempts to compute the distance between two points $(x1, y1)$ and $(x2, y2)$ on the Cartesian plane:

```
1  import math
2
3
4  def dist(x1, y1, x2, y2):
5    '''
6    Returns the distance between (x1, y1) and (x2, y2)
7
8    dist: Float Float Float Float => Float
9
10   Examples:
11      dist(0.0, 0.0, 0.0, 0.0) => 0.0
12      dist(0.0, 0.0, -3.0, 4.0) => 5.0
13   '''
14   return sqrt((x1-x2)**2 + (y1-y2)**2)
```

True or False: The above code is correct.

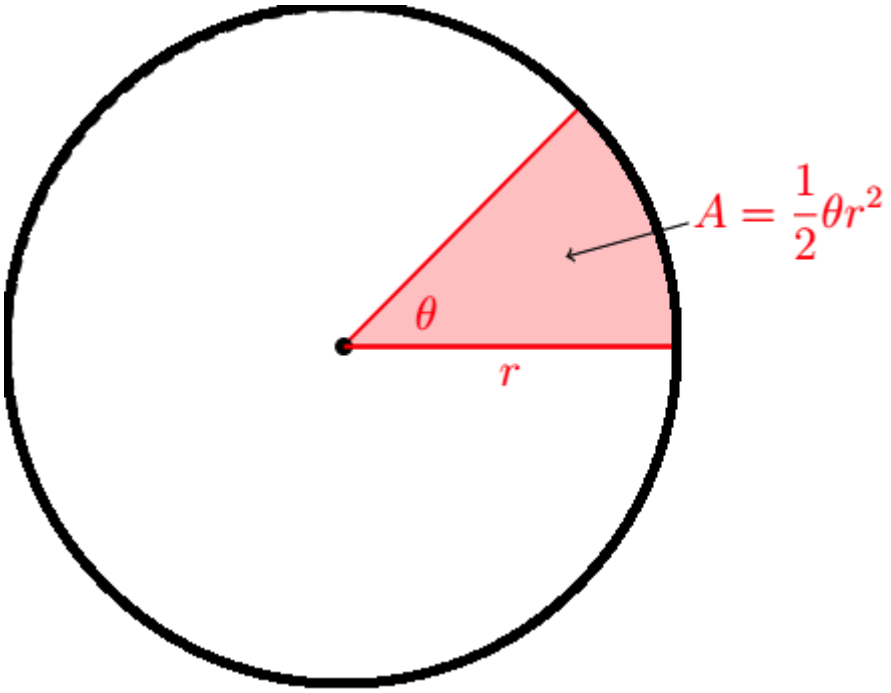
- ◯ True
- ◯ False

Concept Check 1.5.7: Sector Area

The formula for the area of a sector of a circle is given by

$$A = \frac{1}{2}r^2\theta$$

where r is the radius of the circle and θ is the angle of a circle measured in radians. Pictorially,



Write a function `sector_area(r, theta)` that consumes the radius of a circle and the angle of the sector and returns the area.

0/3 points
Attempts: 0 / Unlimited

```
1  def sector_area(r, theta):
2      '''
3      Returns the area of a sector of a circle
4      given the radius r and angle theta
5
6      sector_area: Float Float  -> Float
7      Requires:
8         0.0 <= r
9         0.0 <= theta <= 2*math.pi
10
11     Example:
12        sector_area(0.0, 0.0) => 0.0
13        sector_area(1.0, math.pi/2) => 0.785398163397
14     '''
15     ##YOUR CODE GOES HERE
16     pass
```

Final Thoughts

In this lesson, we discussed our first built-in module, namely the `math` module. We discussed some of the functions and constants therein and how to use them.

This concludes the lessons portion of our first module. We discussed the basics of Python, including variables, arithmetical expressions, and functions, and we discussed how we need to modify our design recipe for our new language: Python. We compared and contrasted functional and imperative programming and why we are going to make the switch to a much higher level language like Python.

All modules will finish with a Wrap-Up Quiz and an assignment testing concepts from the module. These are meant to help you to learn and understand the material more deeply and give the opportunity for self-reflection. In the next module, we will discuss our first control statements, namely `if` statements. With these, we will be able to implement recursion in Python and program most of what you could program in the previous course only now in Python.