CPSC 103 - Introduction to Systematic Program Design

Module 01. Programming in Python

Prof. Karina Mochetti

2020.W2

Announcements

What is new!

http://www.cs.ubc.ca/~mochetti/CPSC103.html

Goals

- Write statements that operate on primitive data including numbers, strings and booleans.
- Write variable definitions and function definitions.
- Write out the step-by-step evaluation of simple statements including function calls.
- Use Jupyter notebooks to run Python code.

Quiz #01 Feedback

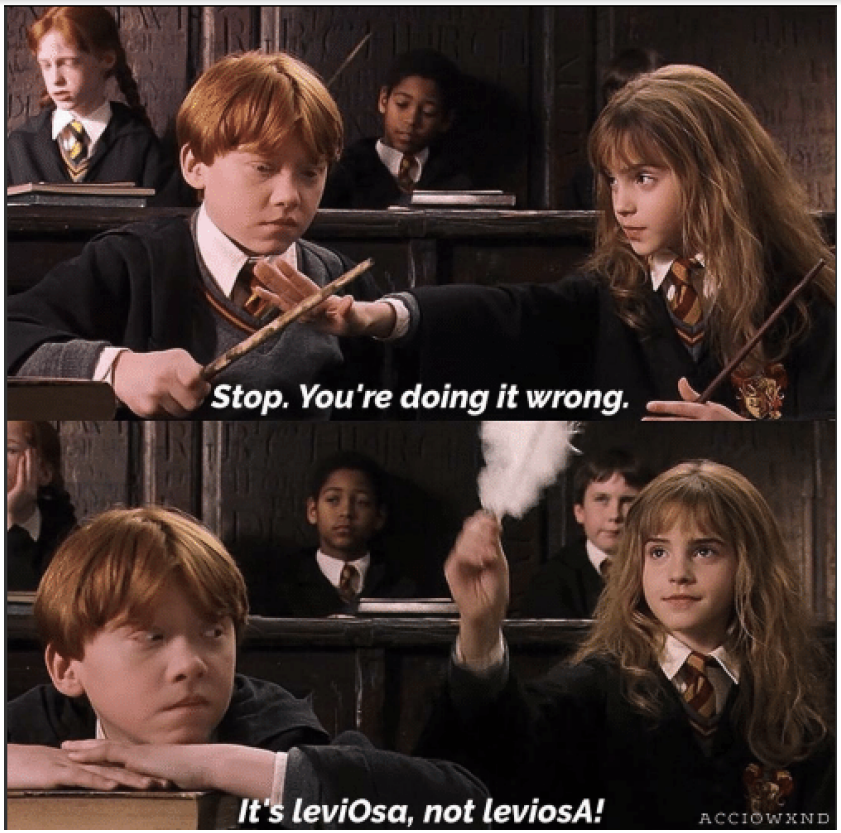
Good job, everyone!

The `sqrt(9)` question was the only one that tripped a lot of people up before the open-ended questions at the end. We'll talk about it a bit later.

- When starting a difficult question, like Question 12 on the quiz, what is the best way to approach the coding, or defining a function?
  This week, we give you very little guidance on this! :(
  Happily, the rest of the term will be all about how to use the problem you're solving and the information it operates on to guide you toward a clear, systematic, correct solution.

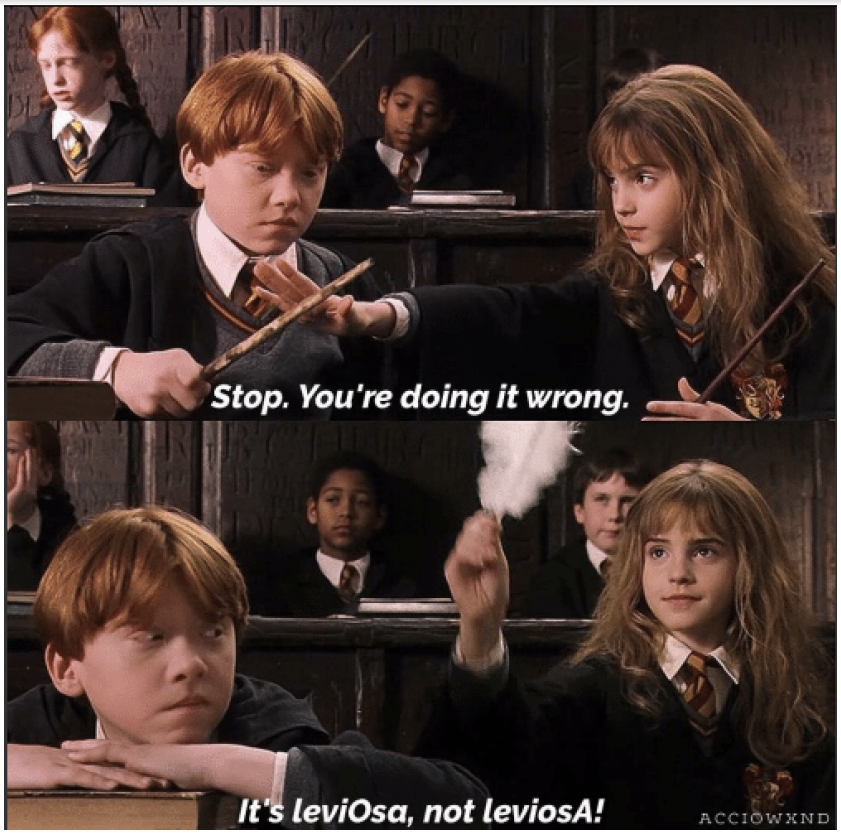
What is a Program

Programs are like magic spells, they can do wonderful things, but they are a tricky process and each detail matters! But when done correctly they can change the world!



What is a Program

Programs are like magic spells, they can do wonderful things, but they are a tricky process and each detail matters! But when done correctly they can change the world!



The not so cool definition is that programs are a sequence of commands gave to the computer for it to execute some calculation.

Example

Jupyter Code: Example

Open the file module-1-intro - Lectures - Karina - Example.ipynb.

Drawing Trees

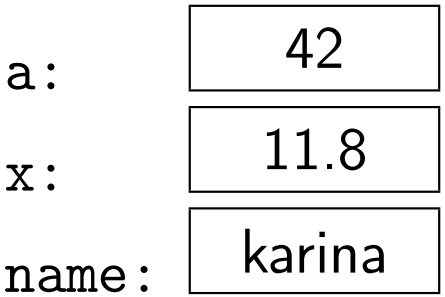
Jupyter Code: Drawing Tress

Open the file module-1-intro - Lectures - Karina - Module 01.ipynb.

Variables

Variable

A variable is a place in memory that can store values. It is like a box that only fits **one** value at a time, every time we change it, we have to remove and discard the previous value.



A value can be a word (**string**), a number (**integer**), a real number (**float**)...

True and False

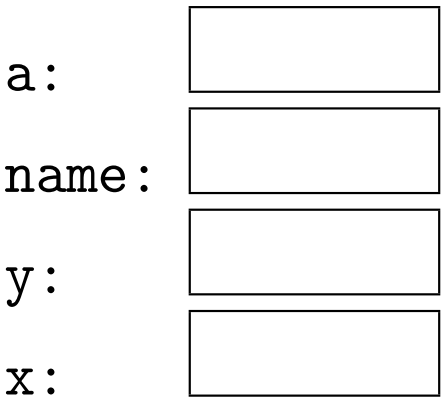
Boolean

A value can also be a Boolean, which can only have two possible values **True** and **False**.

Operations

Assignment

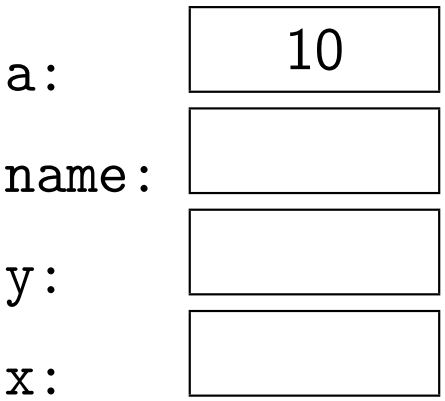
```
a = 10                  # put 10 in variable a
name = "karina"         # put "karina" in variable name
y = 66                  # put 66 in variable y
x = y                   # copy value stored in y in x
x = "y"                 # put "y" in variable x
```



Operations

Assignment

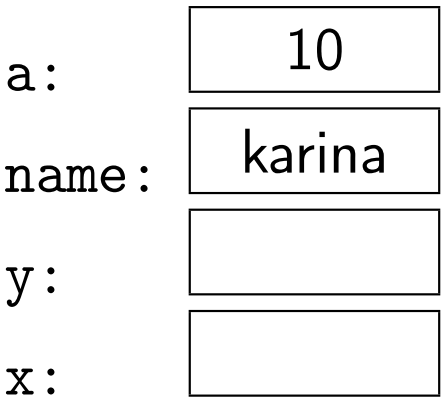
```
a = 10                  # put 10 in variable a
name = "karina"         # put "karina" in variable name
y = 66                  # put 66 in variable y
x = y                   # copy value stored in y in x
x = "y"                 # put "y" in variable x
```



Operations

Assignment

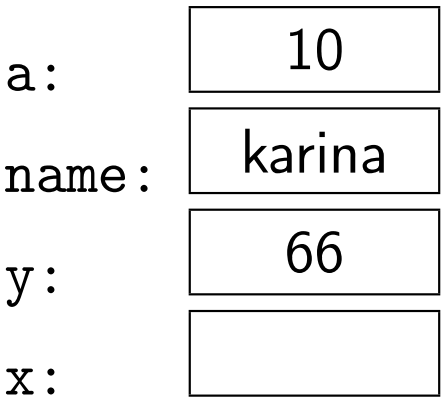
```
a = 10                  # put 10 in variable a
name = "karina"         # put "karina" in variable name
y = 66                  # put 66 in variable y
x = y                   # copy value stored in y in x
x = "y"                 # put "y" in variable x
```



Operations

Assignment

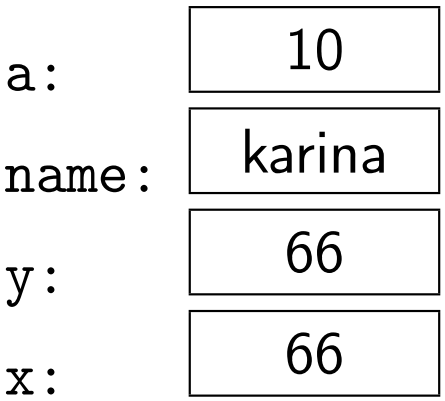
```
a = 10                  # put 10 in variable a
name = "karina"         # put "karina" in variable name
y = 66                  # put 66 in variable y
x = y                   # copy value stored in y in x
x = "y"                 # put "y" in variable x
```



Operations

Assignment

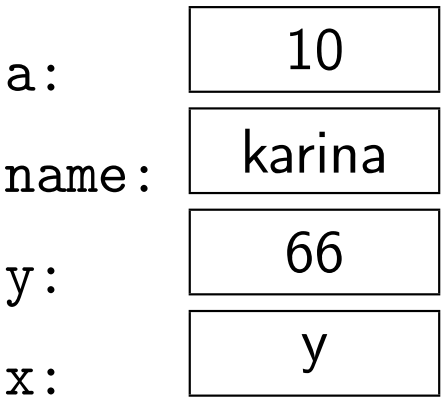
```
a = 10                  # put 10 in variable a
name = "karina"         # put "karina" in variable name
y = 66                  # put 66 in variable y
x = y                   # copy value stored in y in x
x = "y"                 # put "y" in variable x
```



Operations

Assignment

```
a = 10                  # put 10 in variable a
name = "karina"         # put "karina" in variable name
y = 66                  # put 66 in variable y
x = y                   # copy value stored in y in x
x = "y"                 # put "y" in variable x
```



Operations

Addition

```
a = 10 + 2    # put 12 in variable a
x = y + z     # add values of y and z and put in x
```

Subtraction

```
a = 10 - 2    # put 8 in variable a
x = y - z     # subtract values of y and z and put in x
```

Multiplication

```
a = 10 * 2    # put 20 in variable a
x = y * z     # multiply values of y and z and put in x
```

Division

```
a = 10 / 2    # put 5 in variable a
x = y / z     # divide value of y by z and put in x
```

Operations

Equal

```
10 == 10  # evaluate to True since 10 is equal to 10
a == b    # evaluate to True if the value of a is
          # equal to the value of b
```

Not Equal

```
10 != 10  # evaluate to False since 10 is equal to 10
a != b    # evaluate to True if the value of a is
          # not equal to the value of b
```

Greater / Greater and Equal

```
a > b   # evaluate to True if the value of a is
        # greater than the value of b
a >= b  # evaluate to True if the value of a is
        # greater than or equal to the value of b
```

Less / Less and Equal

```
a < b   # evaluate to True if the value of a is
        # less than the value of b
a <= b  # evaluate to True if the value of a is
        # less than or equal to the value of b
```

Example

What is the difference between the codes? What will Jupyter show?

Code 1

```
x = 9
sqrt(9)
```

Code 2

```
x = 9
sqrt(x)
```

Code 3

```
x = 9
x = sqrt(x)
```

Code 4

```
x = 9
x == sqrt(x)
```

Example

What is the difference between the codes? What will Jupyter show?

Code 1

```
x = 9
sqrt(9)
```

Variable x is uninvolved on the calculation. It produces 3, but does nothing with it!

Jupyter will show 3.0.

Example

What is the difference between the codes? What will Jupyter show?

Code 2

```
x = 9
sqrt(x)
```

Evaluates x to 9 so sqrt is called with 9, just like last time. Again, it does nothing to x, produces 3, but does nothing with it!

Jupyter will show 3.0.

Example

What is the difference between the codes? What will Jupyter show?

Code 3

```
x = 9
x = sqrt(x)
```

Evaluates x on the right side to 9, getting 3 as the square root and puts that into the variable shown on the left (replacing, therefore, the value of x).

Jupyter will show nothing.

Example

What is the difference between the codes? What will Jupyter show?

Code 4

```
x = 9
x == sqrt(x)
```

It checks if the value of its left side (x) equals the value of its right side (sqrt(x)). But it doesn't change anything's value!

Jupyter will show False.

Tracing a Code

```
a = 1
b = a + 10
a = a + 10
a == b
b = a + b
a = 100
a + b
```

Tracing a Code

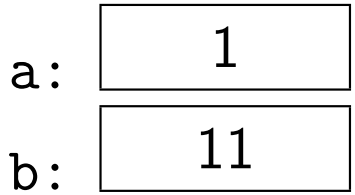
```
a = 1
b = a + 10
a = a + 10
a == b
b = a + b
a = 100
a + b
```

a: 1

After this line, in the computer's "memory", we have the variable a with the value 1.

Tracing a Code

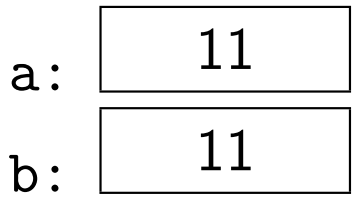
```
a = 1
b = a + 10
a = a + 10
a == b
b = a + b
a = 100
a + b
```



To evaluate a + 10, the computer checks a's value in memory. It's 1. So, a + 10 is 11. We create a new variable in memory called b with the value 11, a still has the same old value of 1.

Tracing a Code

```
a = 1
b = a + 10
a = a + 10
a == b
b = a + b
a = 100
a + b
```



This looks strange but follows about the same rules. To evaluate a + 10, we look up a in memory. Its value is 1. So, a + 10 is 11. We don't create a NEW variable; we just change a's value in memory to 11. a has the value 11; b still has the value 11 as well.

Tracing a Code

```
a = 1
b = a + 10
a = a + 10
a == b
b = a + b
a = 100
a + b
```

a:	11
b:	11

What does this evaluate to? a's and b's values are both 11; so, this evaluates to True BUT, it isn't assignment (the single = sign); so, it doesn't change a's value or b's.

Tracing a Code

```
a = 1
b = a + 10
a = a + 10
a == b
b = a + b
a = 100
a + b
```

a:	11
b:	22

a + b is 22. We change b's value in memory to 22. a has the value 11; b has the value 22.

Tracing a Code

```
a = 1
b = a + 10
a = a + 10
a == b
b = a + b
a = 100
a + b
```

a:	100
b:	22

We change a's value to 100. a has the value 100; be has the value 22.

Tracing a Code

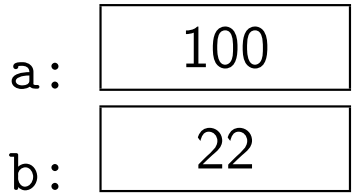
```
a = 1
b = a + 10
a = a + 10
a == b
b = a + b
a = 100
a + b
```

a:	100
b:	22

What does this evaluate to? a + b is 100 + 22, which is 122.

Tracing a Code

```
a = 1
b = a + 10
a = a + 10
a == b
b = a + b
a = 100
a + b
```



We can also use http://www.pythontutor.com/visualize.html#mode=edit to trace code.

Questions 1 - 8

Questions?

Ask CPSC 103

http://www.cs.ubc.ca/~mochetti/askCPSC103.html

Condition

if

```
if <statement>:
    <actions>
```

The statement must evaluate to either True or False. There must be one action per line and all actions with a space before them will be executed only if the statement evaluates to True.

```
if a == b:
    return True
```

Details that matter:

space after **if**

: at the end of the line

all **actions** with the same space before them.

Condition

elif

```
if <statement1>:
    <actions1>
elif <statement2>:
    <actions2>
```

If the first statement evaluates to True, it will execute all actions in the first block and skip all other statements and actions.
If the first statement evaluates to False, it will execute all actions in the second block.

```
if a == b:
    return "equal"
elif a < b:
    return "less"
elif a > b:
    return "greater"
```

else

```
if <statement1>:
    <actions1>
else:
    <actions2>
```

If the all if and elif statements evaluate to False, it will execute all actions in the else block.

```
if a == 0:
    return "equal"
else:
    return "different"
```

Check Sign

Jupyter Code: Check Sign

Open the file module-1-intro - Lectures - Karina - Module 01.ipynb.

Functions

A function is a block of code which only runs when it is called and returns values as a result.

- **Defining** a function is like building a machine, it doesn't do anything. Building a donut-maker doesn't make donuts, just like defining a function in Python doesn't do the thing.
- **Calling** a function is like running the machine and that does the thing. Running a donut-maker makes donuts, just like calling a function in Python does the thing.

Functions

A function is a block of code which only runs when it is called.

- We build the machine once but run it as many times as we need. We define a function once, but call it as many times as we need.
- rectangle(), above(), sqrt() are all example of functions defined by Python.

Parameters and Returns

You can pass values into a function (inputs or parameters) and it can return values as a result (outputs or returns).

- Our donut-maker will make different kinds of donuts if we give it different batter and toppings. Similarly, our functions behave differently if we supply different values for its inputs.
- Our donut-maker produces an output: donuts! Similarly, our functions return results.

Repeat String

Jupyter Code: Repeating a String

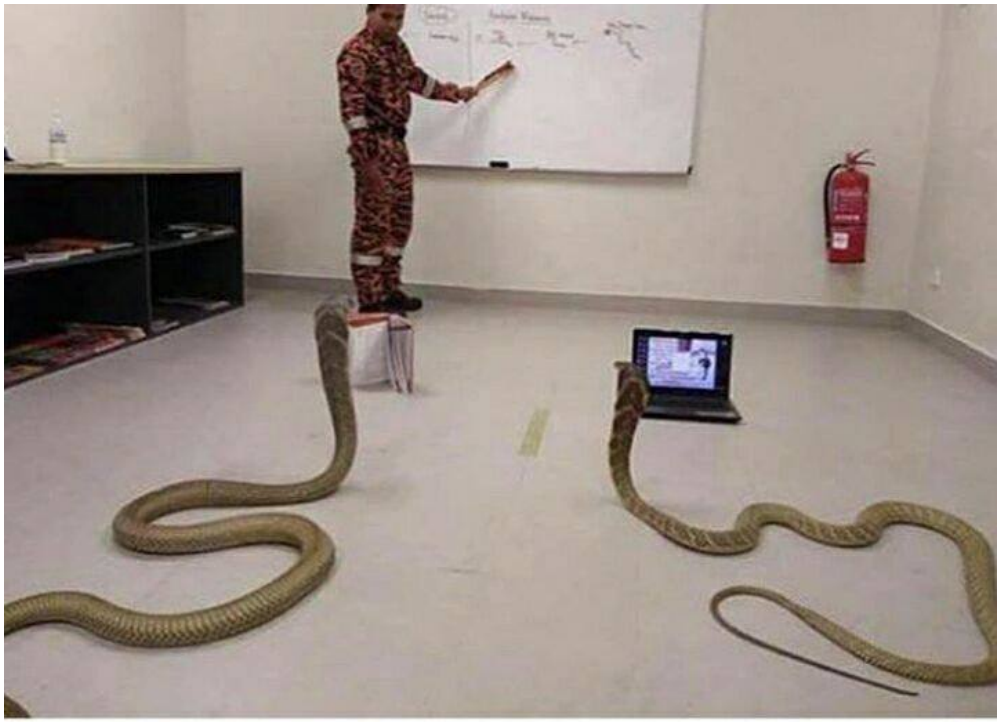
Open the file module-1-intro - Lectures - Karina - Module 01.ipynb.

Questions 9 - 16

End of Module

Ask CPSC 103

http://www.cs.ubc.ca/~mochetti/askCPSC103.html



Teaching python programming 😂😂

Exercise

Let's create a function that make trees of different sizes. We will use the height as input and draw a rectangle with a triangle on top that have this exact height.

Exercise: Solution

Let's create a function that make trees of different sizes. We will use the height as input and returns a rectangle with a triangle on top that have this exact height.

```
def make_tree(height):
  triangle_height = 0.80 * height
  crown = triangle(40, triangle_height, "solid", "green")
  rectangle_height = 0.20 * height
  trunk = rectangle(15, rectangle_height, "solid", "brown")
  return above(crown, truck)

make_tree(100)
```

Exercise: Solution

Let's create a function that make trees of different sizes. We will use the height as input and returns a rectangle with a triangle on top that have this exact height.

```
def make_tree(height):
    triangle_height = 0.80 * height
    crown = triangle(40, triangle_height, "solid", "green")
    rectangle_height = 0.20 * height
    trunk = rectangle(15, rectangle_height, "solid", "brown")
    return above(crown, truck)

make_tree(100)
```

This defines a function, its name should be lowercase with underscores and should also be meaningful.

Exercise: Solution

Let's create a function that make trees of different sizes. We will use the height as input and returns a rectangle with a triangle on top that have this exact height.

```
def make_tree(height):
  triangle_height = 0.80 * height
  crown = triangle(40, triangle_height, "solid", "green")
  rectangle_height = 0.20 * height
  trunk = rectangle(15, rectangle_height, "solid", "brown")
  return above(crown, truck)

make_tree(100)
```

The variable triangle_height has 80% of the value of the input (variable height).

Exercise: Solution

Let's create a function that make trees of different sizes. We will use the height as input and returns a rectangle with a triangle on top that have this exact height.

```
def make_tree(height):
    triangle_height = 0.80 * height
    crown = triangle(40, triangle_height, "solid", "green")
    rectangle_height = 0.20 * height
    trunk = rectangle(15, rectangle_height, "solid", "brown")
    return above(crown, truck)

make_tree(100)
```

The tree crown is a green and solid triangle with height equals triangle_height.

Exercise: Solution

Let's create a function that make trees of different sizes. We will use the height as input and returns a rectangle with a triangle on top that have this exact height.

```
def make_tree(height):
    triangle_height = 0.80 * height
    crown = triangle(40, triangle_height, "solid", "green")
    rectangle_height = 0.20 * height
    trunk = rectangle(15, rectangle_height, "solid", "brown")
    return above(crown, truck)

make_tree(100)
```

The variable rectangle_height has 20% of the value of the input (variable height).

Exercise: Solution

Let's create a function that make trees of different sizes. We will use the height as input and returns a rectangle with a triangle on top that have this exact height.

```
def make_tree(height):
    triangle_height = 0.80 * height
    crown = triangle(40, triangle_height, "solid", "green")
    rectangle_height = 0.20 * height
    trunk = rectangle(15, rectangle_height, "solid", "brown")
    return above(crown, truck)

make_tree(100)
```

The tree trunk is a brown and solid rectangle with height equals rectangle_height.

Exercise: Solution

Let's create a function that make trees of different sizes. We will use the height as input and returns a rectangle with a triangle on top that have this exact height.

```
def make_tree(height):
  triangle_height = 0.80 * height
  crown = triangle(40, triangle_height, "solid", "green")
  rectangle_height = 0.20 * height
  trunk = rectangle(15, rectangle_height, "solid", "brown")
  return above(crown, truck)

make_tree(100)
```

The tree is the crown above the trunk and we use the above function to achieve that.

Exercise: Solution

Let's create a function that make trees of different sizes. We will use the height as input and returns a rectangle with a triangle on top that have this exact height.

```
def make_tree(height):
  triangle_height = 0.80 * height
  crown = triangle(40, triangle_height, "solid", "green")
  rectangle_height = 0.20 * height
  trunk = rectangle(15, rectangle_height, "solid", "brown")
  return above(crown, truck)

make_tree(100)
```

This calls the function and creates a tree with total height 100, with a 80 crown and 20 trunk.