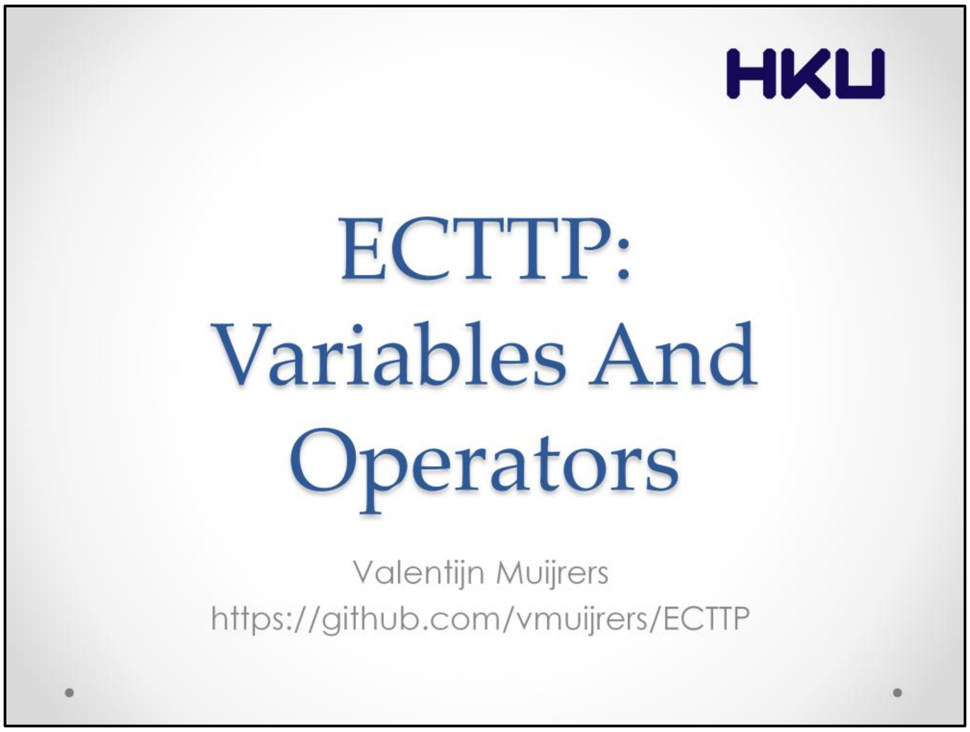
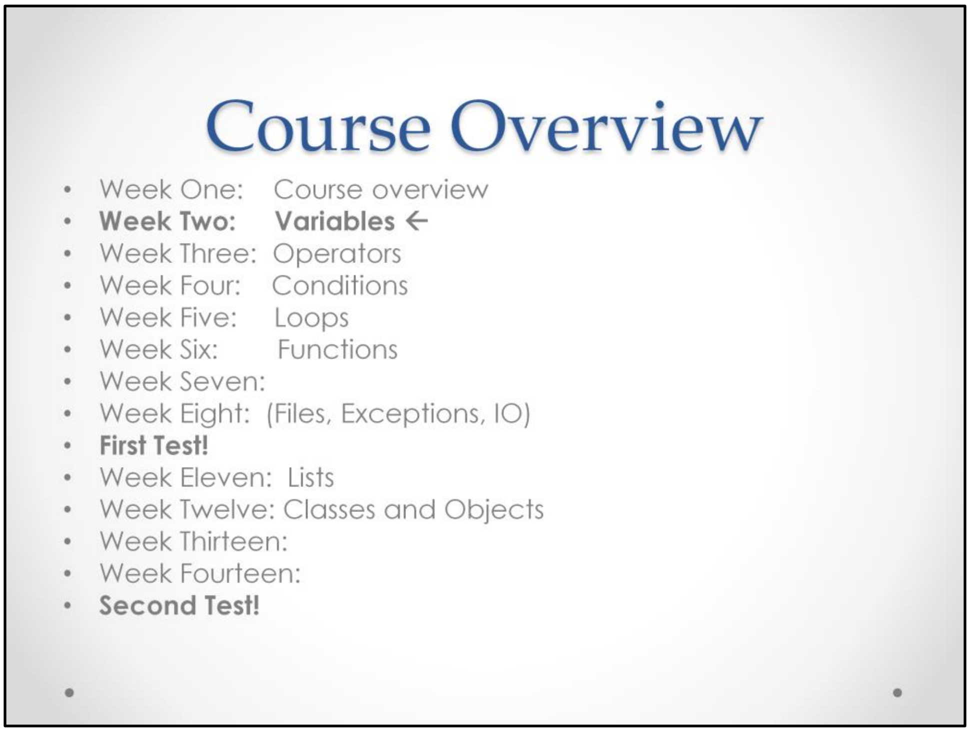


HKU

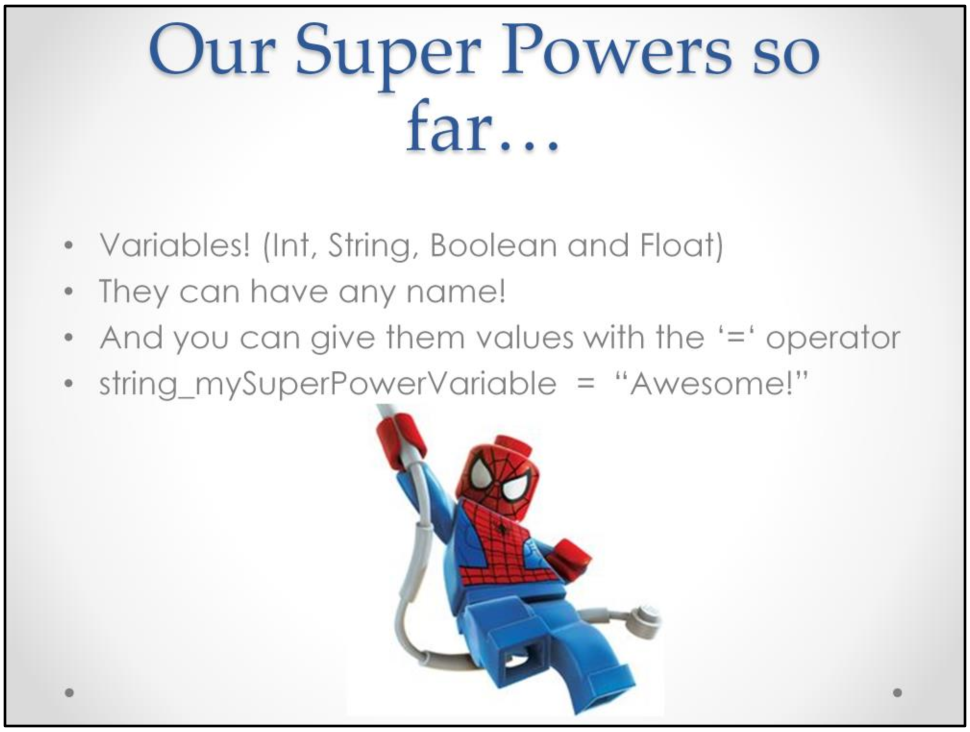
ECTTP: Variables And Operators

Valentijn Muijrers
https://github.com/vmuijrers/ECTTP



Course Overview

- Week One: Course overview
- **Week Two: Variables ←**
- Week Three: Operators
- Week Four: Conditions
- Week Five: Loops
- Week Six: Functions
- Week Seven:
- Week Eight: (Files, Exceptions, IO)
- **First Test!**
- Week Eleven: Lists
- Week Twelve: Classes and Objects
- Week Thirteen:
- Week Fourteen:
- **Second Test!**



Our Super Powers so far…

- Variables! (Int, String, Boolean and Float)
- They can have any name!
- And you can give them values with the '=' operator
- string_mySuperPowerVariable = "Awesome!"

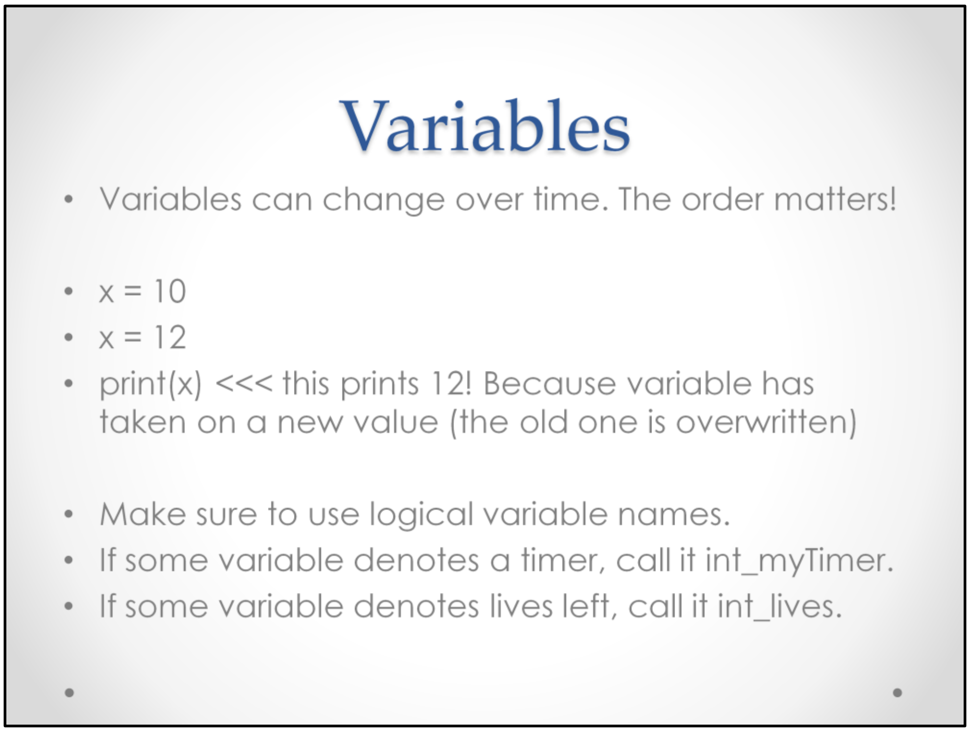
Constants

- Constants are fixed values which are always the same. 10 is always equal to 10.

- Numeric constants are all of the numbers.
- String constants can also be created if you use single quote marks

- print('hello world')
- print 122

- Constants can be assigned to variables

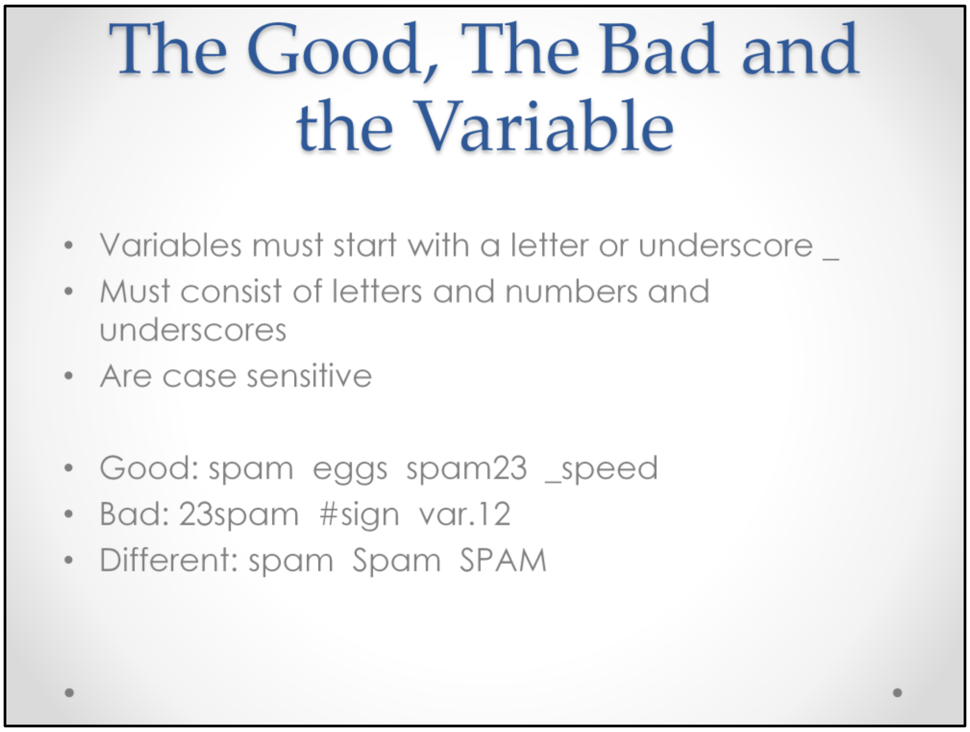


Variables

- Variables can change over time. The order matters!

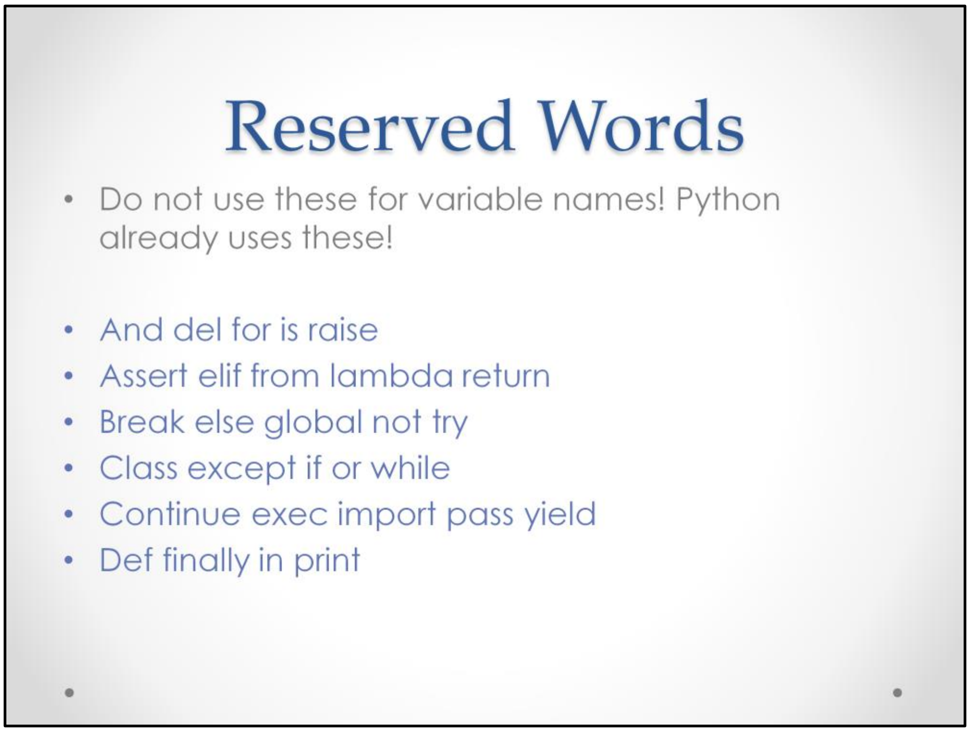
- x = 10
- x = 12
- print(x) <<< this prints 12! Because variable has taken on a new value (the old one is overwritten)

- Make sure to use logical variable names.
- If some variable denotes a timer, call it int_myTimer.
- If some variable denotes lives left, call it int_lives.


The Good, The Bad and the Variable

- Variables must start with a letter or underscore _
- Must consist of letters and numbers and underscores
- Are case sensitive

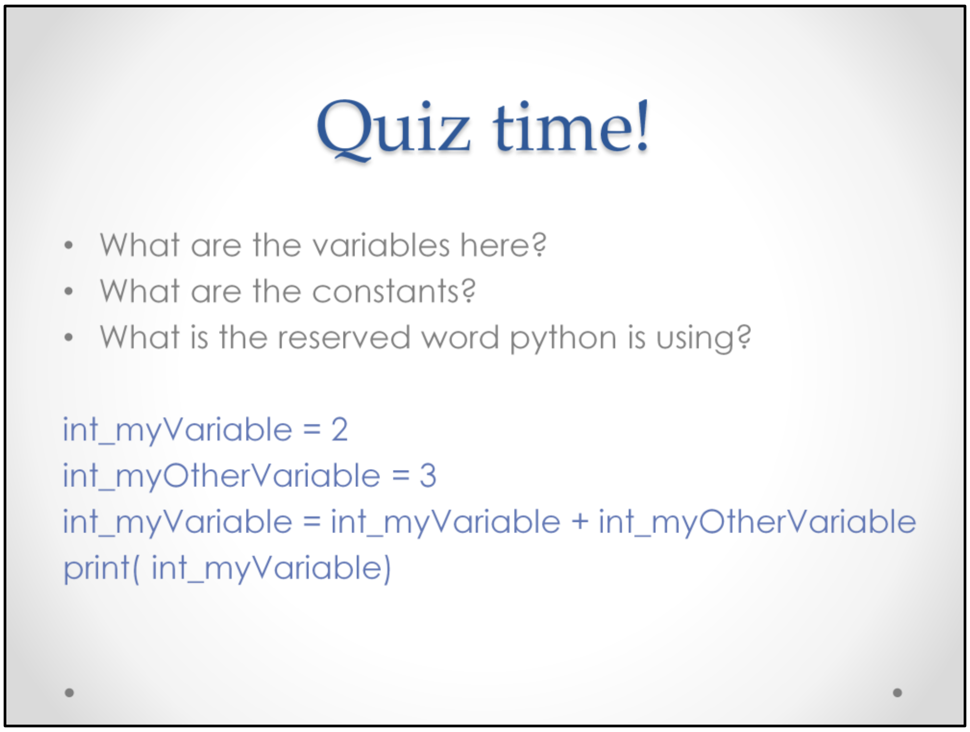
- Good: spam eggs spam23 _speed
- Bad: 23spam #sign var.12
- Different: spam Spam SPAM



Reserved Words

- Do not use these for variable names! Python already uses these!

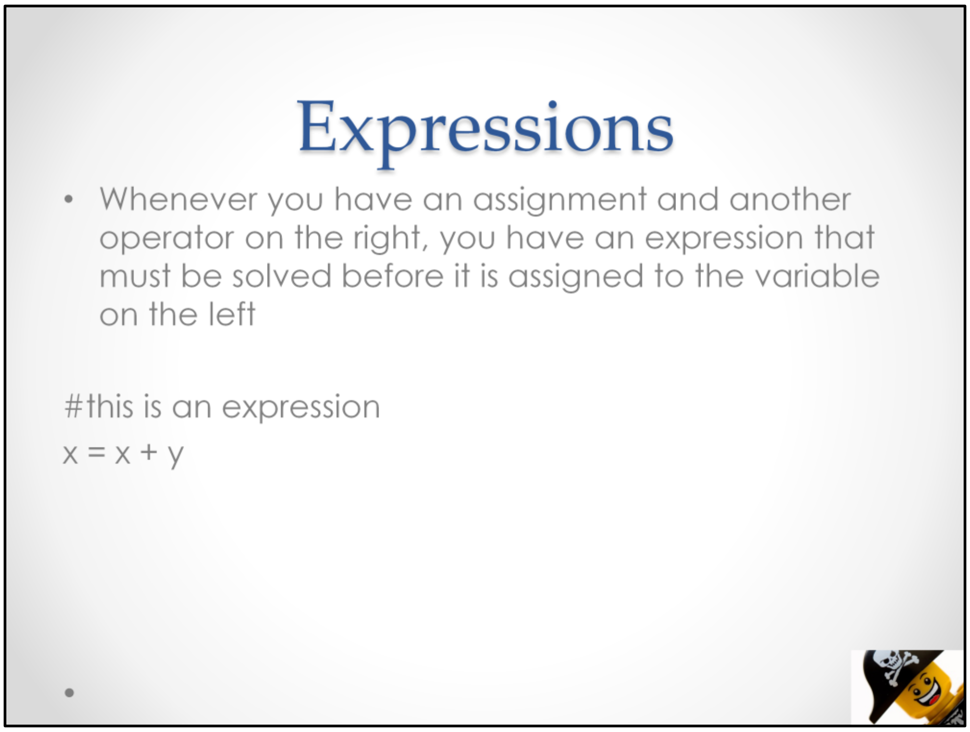
- And del for is raise
- Assert elif from lambda return
- Break else global not try
- Class except if or while
- Continue exec import pass yield
- Def finally in print



Quiz time!

- What are the variables here?
- What are the constants?
- What is the reserved word python is using?

```
int_myVariable = 2
int_myOtherVariable = 3
int_myVariable = int_myVariable + int_myOtherVariable
print( int_myVariable)
```



Expressions

- Whenever you have an assignment and another operator on the right, you have an expression that must be solved before it is assigned to the variable on the left

```
#this is an expression
x = x + y
```

-

Mathematical operators

Math is fun! (or weird)

#The '/' operator is used for division, but….

x = 8 / 3

print (x) << this prints 2

If you divide any **whole numbers**

Together and get a remainder,

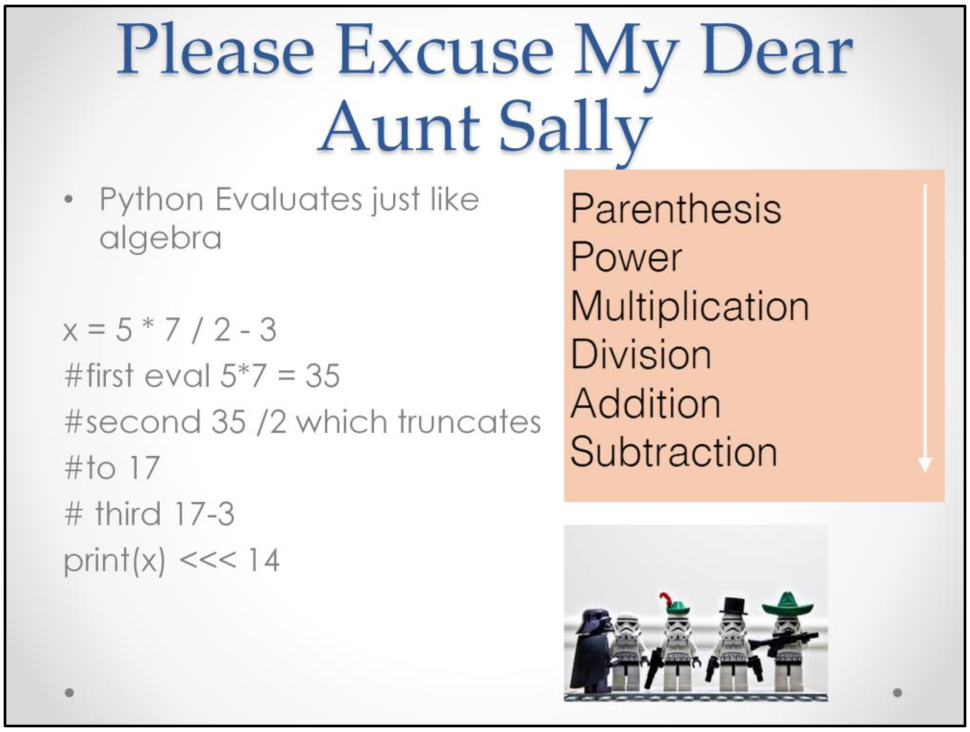
Python gives you a whole number and **truncates the decimal**.

Mathematical operators

```
#use a float instead!
x = 8.0 / 3
print (x) <<< 2.666666

#Another operator is multiply!
x = 5 * 8

#Another operator is subtract
x = 10 -12
```



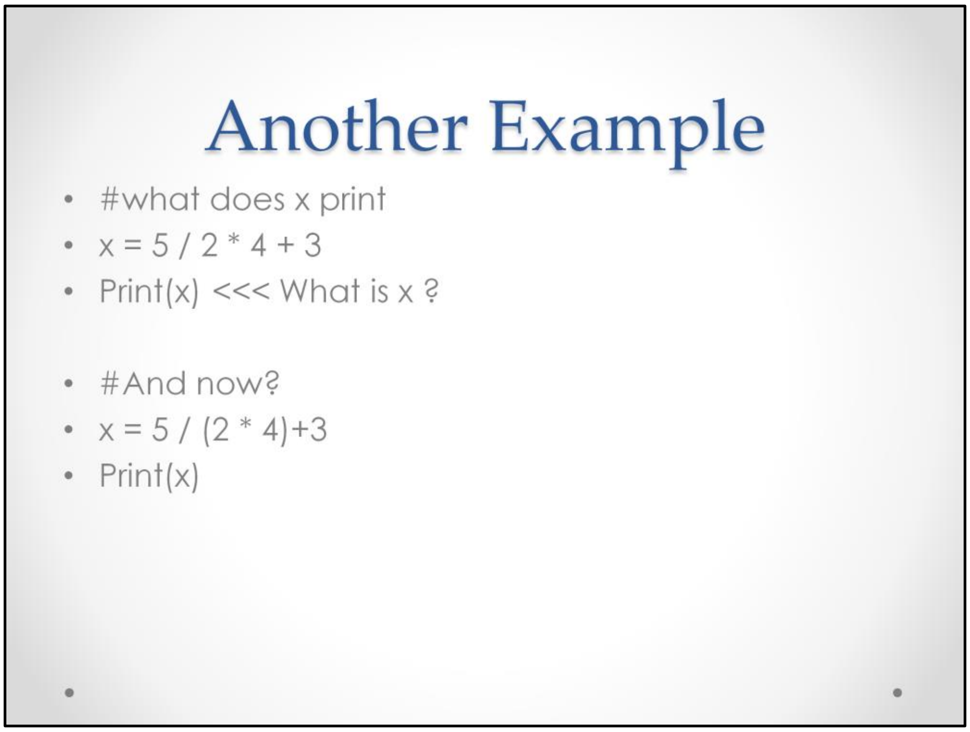
Please Excuse My Dear Aunt Sally

- Python Evaluates just like algebra

```
x = 5 * 7 / 2 - 3
#first eval 5*7 = 35
#second 35 /2 which truncates
#to 17
# third 17-3
print(x) <<< 14
```

Parenthesis
Power
Multiplication
Division
Addition
Subtraction

In python version 3, the multiplication is equal to the division, see:
https://docs.python.org/3/reference/expressions.html
For more details

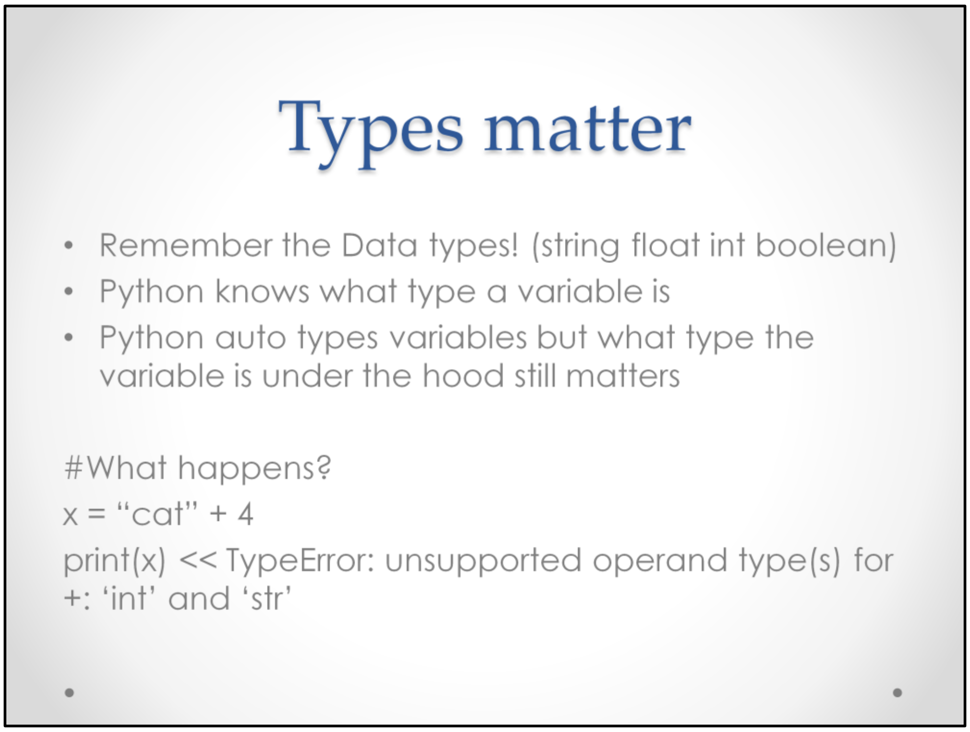


Another Example

- #what does x print
- x = 5 / 2 * 4 + 3
- Print(x) <<< What is x ?

- #And now?
- x = 5 / (2 * 4)+3
- Print(x)

X prints 11 => 5/2 = 2 *4 =8 +3 = 11 in Python 3.x the right answer is 11
X prints 3 => 4+3 = 7......5/ 7 =0 …0+3= 3 in Python 3.x the right answer is 3



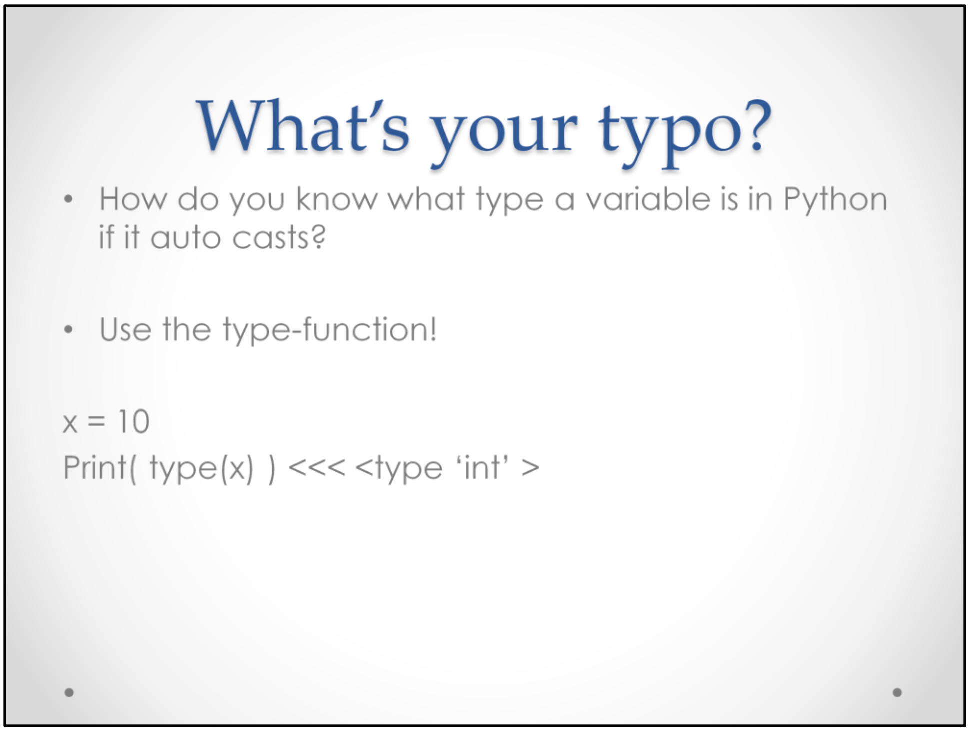
Types matter

- Remember the Data types! (string float int boolean)
- Python knows what type a variable is
- Python auto types variables but what type the variable is under the hood still matters

#What happens?

x = “cat” + 4

print(x) << TypeError: unsupported operand type(s) for +: ‘int’ and ‘str’



What’s your typo?

- How do you know what type a variable is in Python if it auto casts?
- Use the type-function!

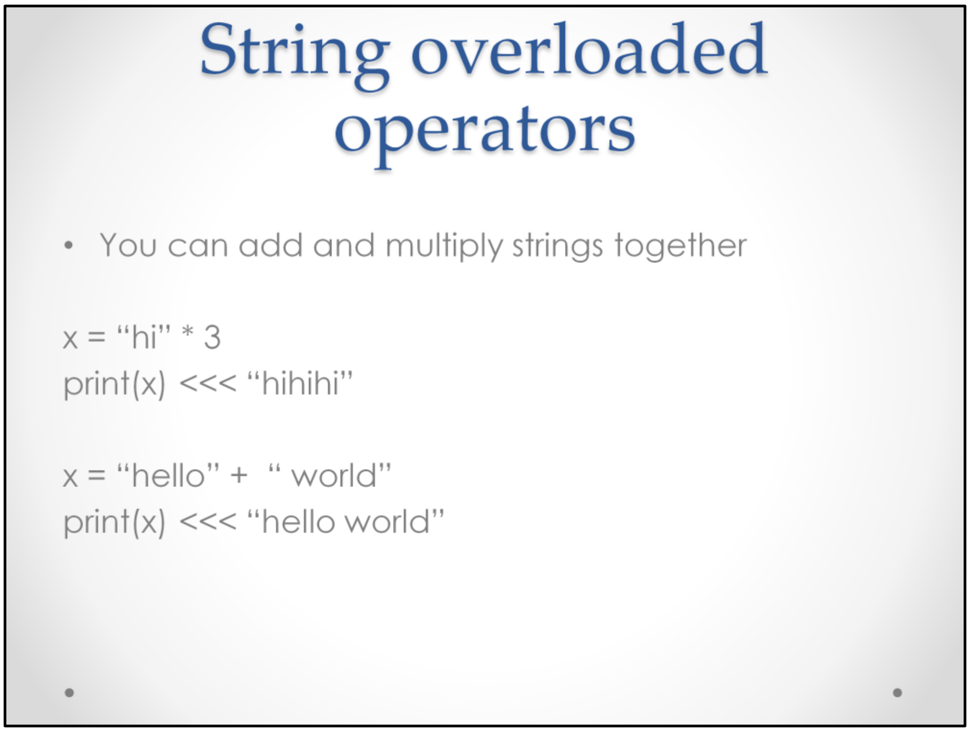
```
x = 10
Print( type(x) ) <<< <type ‘int’ >
```

Type Casting

- What if you have a string and need an int?
- Use the **int()** function!

```
x = "10"
print(type(x) )
<type 'str'>
x = int ( x )
print (type( x ))
<type ' int' >
```

If you need a string use **str(** x **)** and **float (** x **)** for a float!

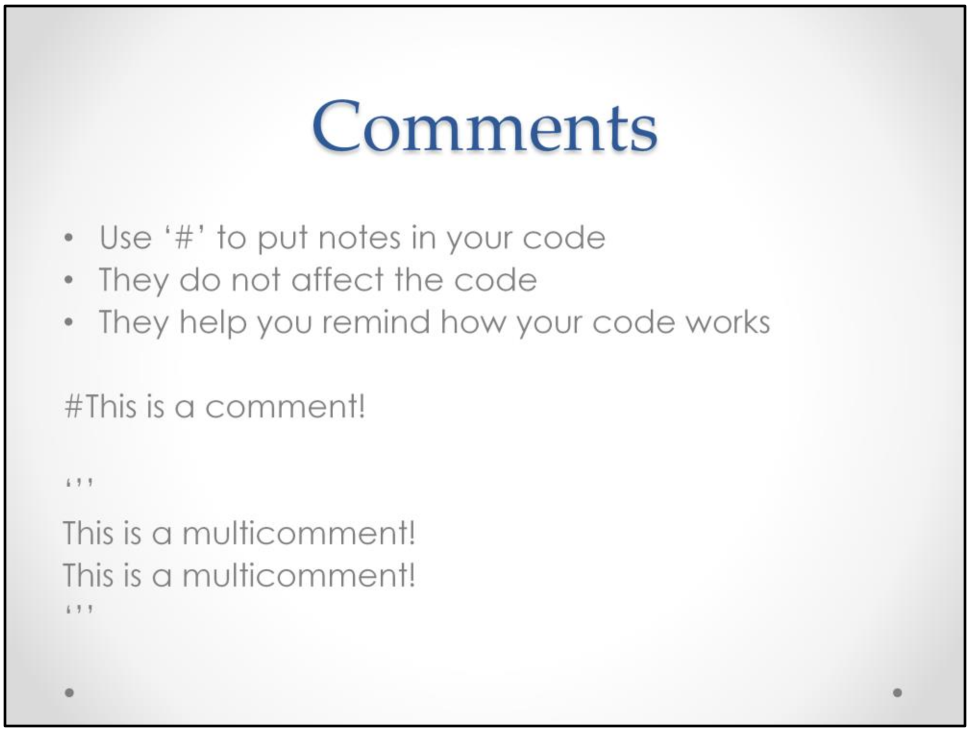


String overloaded operators

- You can add and multiply strings together

```
x = “hi” * 3
print(x) <<< “hihihi”

x = “hello” +  “ world”
print(x) <<< “hello world”
```

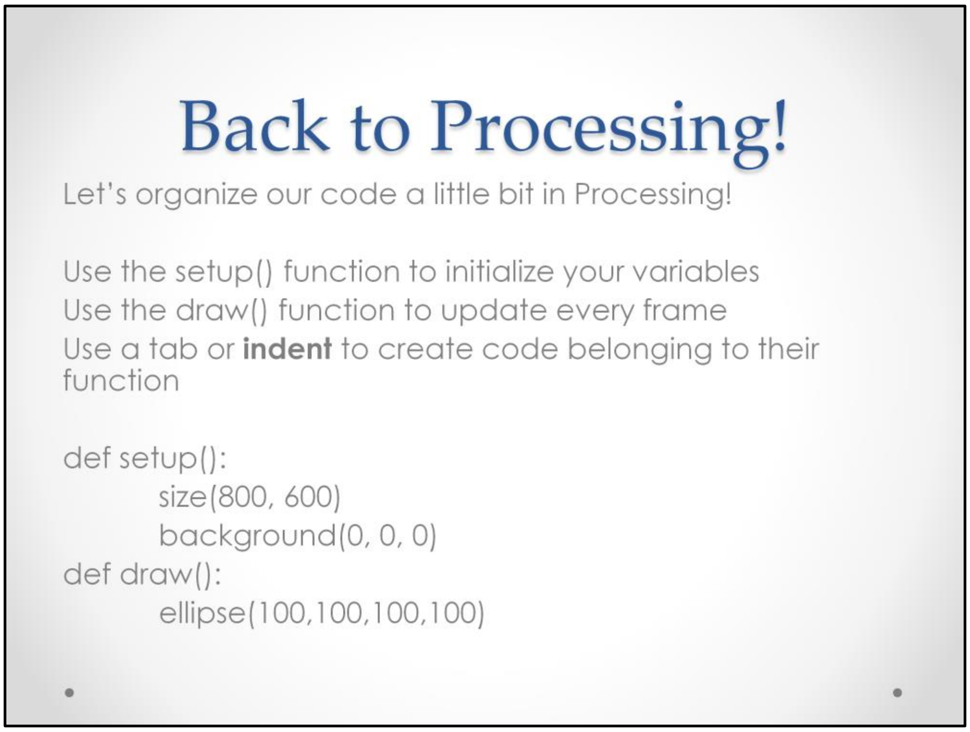


Comments

- Use ‘#’ to put notes in your code
- They do not affect the code
- They help you remind how your code works

```
#This is a comment!

‘’’
This is a multicomment!
This is a multicomment!
‘’’
```


Back to Processing!

Let’s organize our code a little bit in Processing!

Use the setup() function to initialize your variables
Use the draw() function to update every frame
Use a tab or **indent** to create code belonging to their function

```
def setup():
        size(800, 600)
        background(0, 0, 0)
def draw():
        ellipse(100,100,100,100)
```

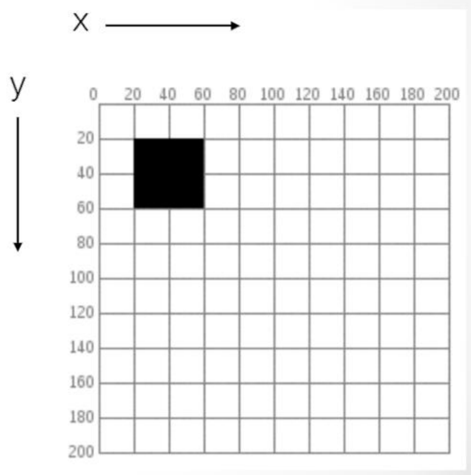
Global Variable

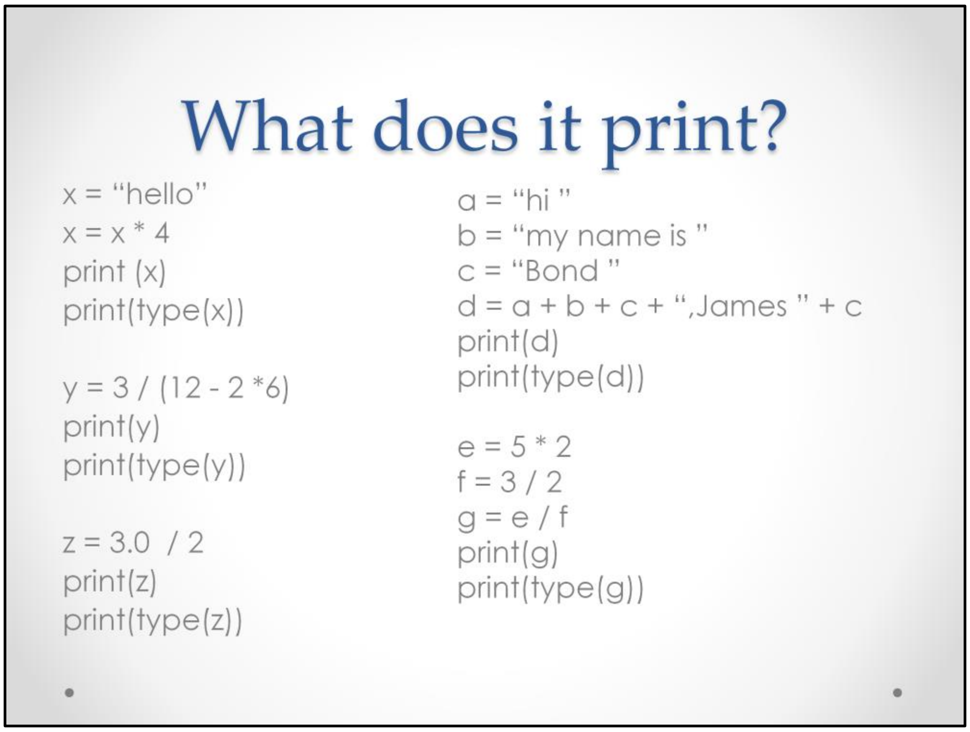
- Use the ‘global’ word before a variable so that it is accessible in every function

```
def setup():
        global x
        x = 10
def draw():
        global x
        x = x + 1
        print ( x )
```

The Origin

- Your grid is in the upper left corner and starts with 0,0
- Use the '**width**' and '**height**' variable to access the size of your screen directly





What does it print?

```
x = "hello"
x = x * 4
print (x)
print(type(x))

y = 3 / (12 - 2 *6)
print(y)
print(type(y))

z = 3.0  / 2
print(z)
print(type(z))
```

```
a = "hi "
b = "my name is "
c = "Bond "
d = a + b + c + ",James " + c
print(d)
print(type(d))

e = 5 * 2
f = 3 / 2
g = e / f
print(g)
print(type(g))
```

Second lab is online

https://github.com/vmuijrers/ECTTP/blob/master/Labs/Lab_2.md

#For examples/tutorials and references!
py.processing.org

#For more practice with python!
codecademy.com