Foundations of Programming

Variables, data and methods

Learning outcomes/key ideas

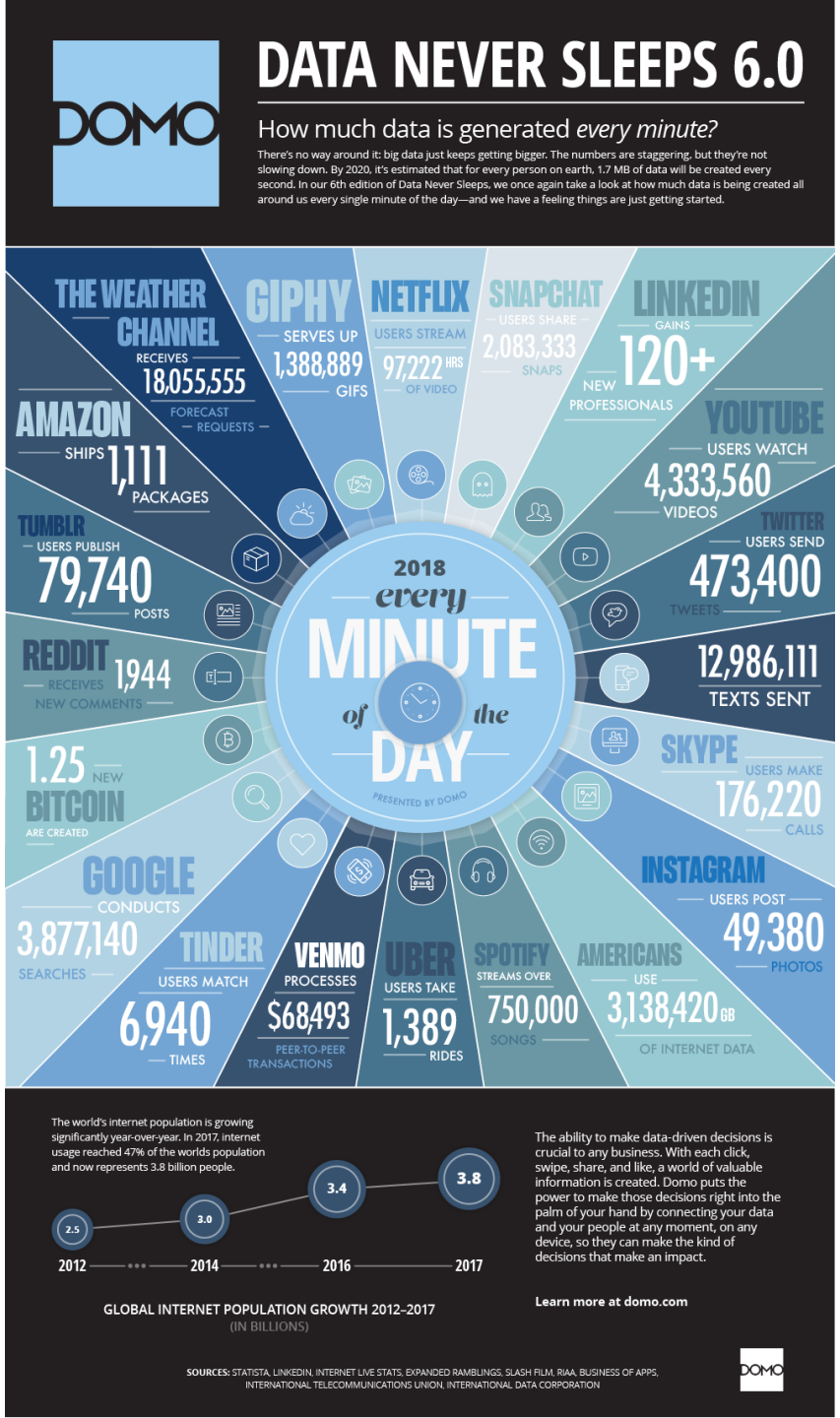
- Describe (and change) the type of data in Python
- Store data in variables using assignment statements
- Describe the difference between printing and returning values
- Call built-in functions in Python
- Load libraries

How much do you like Picobot?

- A. A lot
- B. Some
- C. Not much
- D. Not at all

How far did you get?

- A. Empty room
- B. Maze
- C. One of the additional rooms and/or Empty room or maze in fewer rules
- D. A general solution for ALL the rooms



BIG Data

"There are 2.5 quintillion bytes of data created each day at our current pace, but that pace is only accelerating with the growth of the Internet of Things (IoT)."
https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read

https://www.domo.com/learn/data-never-sleeps-6

Python Data Types

Numeric

Name	Example	What is it?
`float`	`3.14`	values with a fractional part
`long`	`10**100`	integers > 2147483647
`int`	`42`	integers <= 2147483647

`bool`	`True`
	`False`
	"Boolean value"
		the results from a comparison:
		==, !=, <, >, <=, >=

Datatypes as genes…

What will these results be? Why?

Dominant

`float` `1.0 / 5`

`long` `10**100 - 10**100`

`int` `1 + 5`

`bool` `41 + True`

Recessive

`2**False == True`

All data in Python has a type

But you can change its type… implicitly (i.e. last slide) or explicitly through casting

```
>>> int(4.2)

>>> float(True)

>>> int(4)/5

>>> int(4/5)

>>> str(42)
```

Python Operators

Precedence

( ) Highest

**

-

* / %

+ -

> < ==

=

Lowest

Caution Level

set equal to =

divide /

remainder %

power **

is equal to ==

as usual: * + > < - ( )

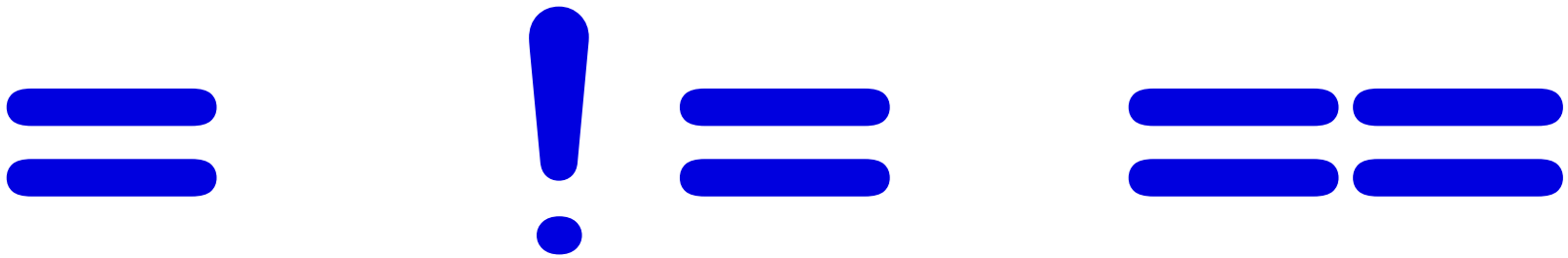
It's not worth remembering all these %+/* things!

I'd go with parentheses over precedence

Key ideas so far

- All data has a type in Python
- You can change the type of data in Python (and sometimes it changes implicitly)
- You can find out the type of a piece of data using the built-in function `type()`

the "equals" operators



This is true – but what is it saying!?

the "equals" operators

=

SET equals

!=

isn't equal to

==

IS equals

I want === !

= *names data*

```
>> x = 41
>> y = x + 1
```

x and y are called “variables”
Don’t confuse them with variables from math
In Python, variables store data

Choosing the right name is more important than I thought.

Inside the machine…

What's happening in python:

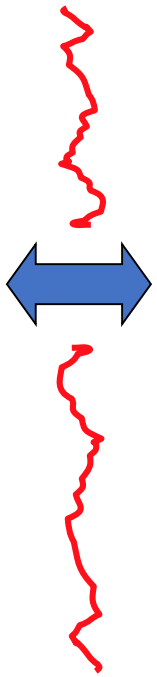
```
x = 41
y = x + 1
```

Inside the machine…

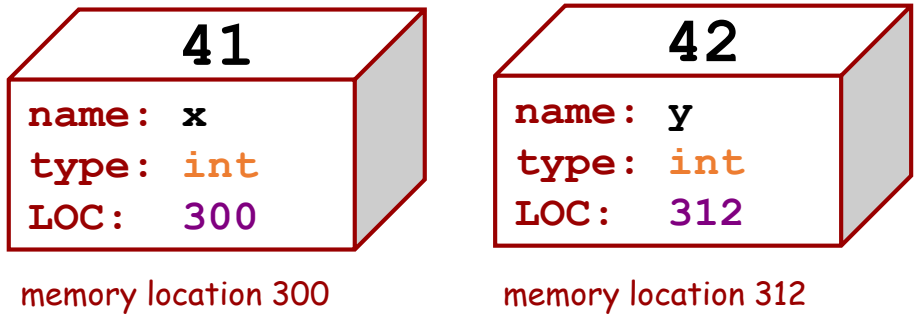
What's happening in python:

```
x = 41
y = x + 1
```

What is happening behind the scenes:



"variables as containers"



Computation

Data Storage

id, del

Memory!

Random Access Memory (RAM)

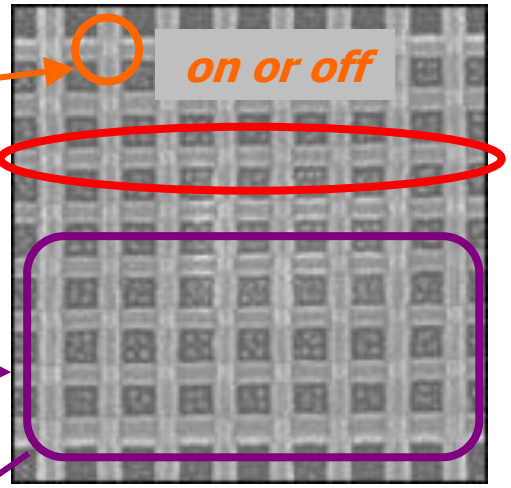
RAM is a long list of memory locations:

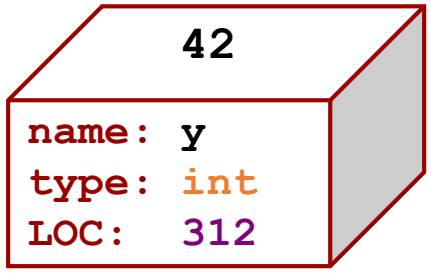
bit = 1 "bucket" of charge

byte = 8 bits

word in 2005 = 4 bytes = 32 bits

word in 2012 = 8 bytes = 64 bits





In Python, an integer uses 4 bytes (plus tax).

Wow! Who knew they make memory out of wicker?

assignment, not equality!

= is an ACTIVE, DIRECTIONAL operator. It means:

"First calculate the value on the right hand side, and then put it into the box labeled with the name from the left hand side (replacing what was there, if necessary)."

It does not test for equality (that's ==).

```
>> x = 41
```
"Put 41 into the box labeled x"

```
>> y = x + 1
```
"Get the value out of x (41), and add 1 to it (42).
Put that value (42) into the box labeled y"

x [ ] y [ ]

Re-naming…!

```
>> x = 41
>> y = x + 1
>> x
41
>> y
42
>> x = x + y
>> x
?? (1)
>> y
??
```

What value is displayed for x at ??(1)?

A. 41
B. 42
C. 83
D. 84

"Find the value in x and add it to the value in y. *Then* place that value back into x, replacing what was there."

x [ ] y [ ]

Re-naming…!

```
>> x = 41
>> y = x + 1
>> x
41
>> y
42
>> x = x + y
>> x
?? (1)
>> y
?? (2)
```

What value is displayed for y at ??(2)?

A. 41
B. 42
C. 83
D. 84

"Find the value in x and add it to the value in y. *Then* place that value back into x, replacing what was there."

x [ ] y [ ]

Re-naming…!

```
>> x = 42
>> y = x
>> x = 101
>> x
??
>> y
??
```

What values are displayed for x and y?

	x	y
A.	42	42
B.	101	42
C.	101	101
D.	None of these	

x 

y 

When in doubt, draw it out!!

Key ideas so far

- All data has a type in Python
- You can change the type of data in Python (and sometimes it changes implicitly)
- You can find out the type of a piece of data using the built-in function `type()`
- Variables store data
- The assignment operator takes the *value* from the right hand side and stores it into the *variable* on the left hand side

Strings

```
>>> "Christine"
'Christine'
>>> 'Christine'
'Christine'
>>> type("Christine")
<class 'str'>
>>> Christine
```

What will idle show if I press enter after the last line?

A. 'Christine'

B. <class 'str'>

C. Error

The `print` function

```
>>> name = "Christine"
>>> name
'Christine'
>>> print(name)
Christine
>>> print("Hello", name)


>>> print("Hello", "name")
```

`print` is a function that is built-in to Python
It takes one or more *arguments* (separated by commas)
It converts these values to Strings and displays them.

Side effects vs. values

```
>>> name = "Christine"
>>> val = print(name)
(1)
>>> val
(2)
```

What will be displayed at (1)?

A. 'Christine'
B. Christine
C. name
D. val
E. Nothing

What will be displayed at (2)?

A. 'Christine'
B. name
C. val
D. Nothing
E. Error

Unlike `print`, some functions return (give back) a value

```
>>> name = "Christine"
>>> numChars = len(name)
>>> numChars
```

What will the value of numChars be?

A. 4
B. 9
C. Something else

Python built-in functions: https://docs.python.org/3/library/functions.html

Other functions have to be loaded from libraries

```
>>> name = "Christine"
>>> numChars = len(name)
>>> numChars
>>> myTweet = "I'm here at SPIS and it's really fun.
I've met lots of cool people and my projects for FoCS are
the best.  I think San Diego is amazing.  Last night I
had some fish tacos and tonight I'm going to have some
sushi."
```

What if I want to shorten this to 140 characters so I can tweet about it?
textwrap to the rescue!

Other functions have to be loaded from libraries

```
>>> name = "Christine"
>>> numChars = len(name)
>>> numChars
>>> myTweet = "I'm here at SPIS and it's really fun.  I've met lots of cool people and my projects for FoCS are the best.  I think San Diego is amazing.  Last night I had some fish tacos and tonight I'm going to have some sushi."
>>> import textwrap
>>> textwrap.shorten(myTweet, 140, placeholder="...")
```

What if I want to shorten this to 140 characters so I can tweet about it?
textwrap to the rescue! https://docs.python.org/3/library/textwrap.html

Key ideas so far

- All data has a type in Python
- You can change the type of data in Python (and sometimes it changes implicitly)
- You can find out the type of a piece of data using the built-in function `type()`
- Variables store data
- The assignment operator takes the *value* from the right hand side and stores it into the *variable* on the left hand side
- Functions are "packaged up" pieces of code that do something (hopefully useful)
- *arguments* (sometimes called *parameters*, more on this later) are data passed in to a function
- Some functions return a value, while others do not
- Python has a few built-in functions, many more in libraries, or you can write your own! (Coming next)

Running code from a file

```
>>> name = "Christine"
>>> numChars = len(name)
>>> numChars
9
>>> print("Hello", name)
Hello Christine
```

I like this code. I want to save it.

To do so I must put it in a file! (Demo)

If I put these lines in a file and run the file, what will Python print?

fileDemo.py

```
name = "Christine"
numChars = len(name)
numChars
print("Hello", name)
```

A. 9
   Hello Christine

B. Hello Christine

C. Nothing

*Function*ing in Python

```
# my own function!
def greeting(personToGreet):
    """ prints a friendly greeting """
    print("Hello", personToGreet)
```

*Function*ing in Python

```
# my own function!
def greeting(personToGreet):
    """ prints a friendly greeting """
    print("Hello " + str(personToGreet))
```

I can load this function by pressing F5.
I can then call it as follows:

```
>>> name = "Christine"
>>> greeting("name")
```

What will the function call to the left print?

A. Hello Christine
B. Hello name
C. Nothing
D. It will cause an error

Flow of Execution

```
# my own function!
def greeting(personToGreet):
    """ prints a friendly greeting """
    print("Hello " + str(personToGreet))
    print("Welcome to SPIS!")

>>> greeting("Eduardo")
```

When you call a function, Python executes the function starting at the first line in its body, and carries out each line in order (though some instructions cause the order to change… more soon)

Common errors…

`return` vs. `print`

```
def greetingReturned(personToGreet):
    """ returns a friendly greeting """
    return "Hello" + str(personToGreet)
```

I can load this function by pressing F5.
I can then call it as follows:

```
>>> name = "Christine"
>>> greetingReturned(name)
```

What will be displayed when I make this function call?

A. Hello Christine
B. Hello name
C. Nothing
D. It will cause an error

`return` vs. `print`

```
def greetingReturned(personToGreet):
    """ returns a friendly greeting """
    return "Hello" + str(personToGreet)

def greeting(personToGreet):
    """ prints a friendly greeting """
    print("Hello " + str(personToGreet))
```

Write code in the shell that can illustrate the difference between these two functions. *Hint: use variables to store the value returned from the function.*

Key ideas so far

- All data has a type in Python
- You can change the type of data in Python (and sometimes it changes implicitly)
- You can find out the type of a piece of data using the built-in function `type()`
- Variables store data
- The assignment operator takes the *value* from the right hand side and stores it into the *variable* on the left hand side
- Functions are "packaged up" pieces of code that do something (hopefully useful)
- *arguments* (sometimes called *parameters*, more on this later) are data passed in to a function
- Some functions return a value, while others do not
- Python has a few built-in functions, many more in libraries, or you can write your own!
- Returning is not the same as printing!