1 Review

Last time:

- The kernel is the program at the "heart" of your computer. It is the first program to start when your computer boots and interfaces between the hardware and software.
- We can interact with the kernel via the OS (GUI) or shell (text terminal/command line).
- Two steps for programming in Python. (1) Write Python code (which is just text) and (2) Use a program called the interpreter to execute the code.
- IDE: allows us to write and execute code using one all-inclusive program (i.e. with a single button click).

2 Python on the Terminal

Demo: you can interact directly with the python interpreter from the command line. Demo print statement, math. (cltr-d to quit).

What's happening: you're writing code in the terminal and sending it the interpreter to execute when you press enter.

Demo: you can also run a .py file from the command line with the python command.

Here, we're invoking the interpreter and telling it to execute the file we're specifying rather than starting an interactive session (before).

This process is not ideal

- Have to save the .py file every time you make a change
- Have to type a command on the terminal (different window) every time you want to run a piece of code
- What if your program outputs something other than text, e.g. plots. Terminal does not handle it elegantly...
- We're going to solve this problem in the next section

3 Jupyter Notebook

Q: What is Anaconda?

- NOTE: Students should have downloaded Anaconda before lecture. Make sure Anaconda is added to PATH!
- Anaconda is a distribution of PYTHON with some additional software.
- One such software we're interested in is Jupyter notebook

- Anaconda is also popular in research because people can share packages with Conda

Q: What is Jupyter notebook?

- Jupyter is an IDE. There are other IDEs!
- **Demo (students follow along):** open Jupyter notebook from the command-line.
- Jupyter starts a server and you interact with it's GUI via a browser. (Not important for you to understand, but that's why you're using a web browser)
- Demo: creating new notebook.
- Jupyter notebook files has for .ipynb (ipython notebook)
- Jupyter is organized by cells. You can run code in each cell independent of other cells.
- Jupyter has code cells and markdown cells where we can type text/math/images
- To run cell: shift-enter
- To enter command mode: press esc. (The cell is surrounded by a green in edit mode and blue in command mode). In command mode "dd" to delete cell, "A" to add cell above, and "B" to add cell below.
- Jupyter notebooks have a kernel. This "kernel" does not have the same meaning as what we discussed last lecture. Think "kernel"="Python interpreter"
- Jupyter default autosave interval: 120 seconds

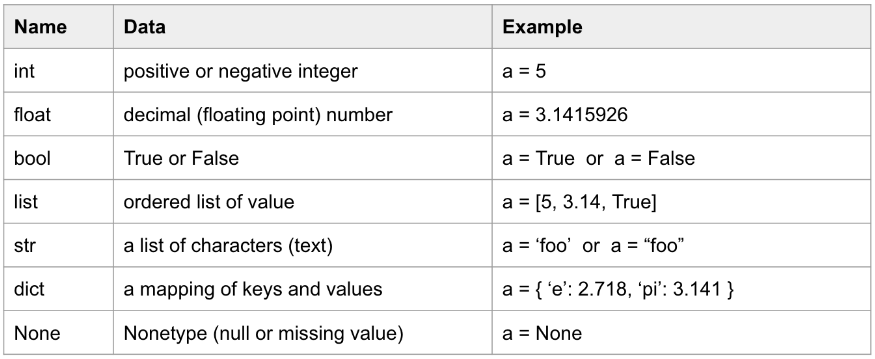
4 Python Intro

Syntax:

- Python is case-sensitive and indentation sensitive
- Comments: hash symbol for single line comment. Triple quotes (”””) surrounding multi-line comment

Variables:

- Containers for storing data we want to keep track of



Name	Data	Example
int	positive or negative integer	a = 5
float	decimal (floating point) number	a = 3.1415926
bool	True or False	a = True or a = False
list	ordered list of value	a = [5, 3.14, True]
str	a list of characters (text)	a = ‘foo’ or a = “foo”
dict	a mapping of keys and values	a = { ‘e’: 2.718, ‘pi’: 3.141 }
None	Nonetype (null or missing value)	a = None

- Demo: We can determine the data type with the type() function

Dynamic typing

- We do not have to specify the type of variable beforehand.
- We can change the type of data that a variables stores easily. **Demo:** $a = 5, a =$‘foo’
- Convenient, but sometimes causes issues when the type of variable is not what you or the compiler expected.
- To create a variable, just assign it a value
- Casting: specifying the type that a variable should be. **Demo:**

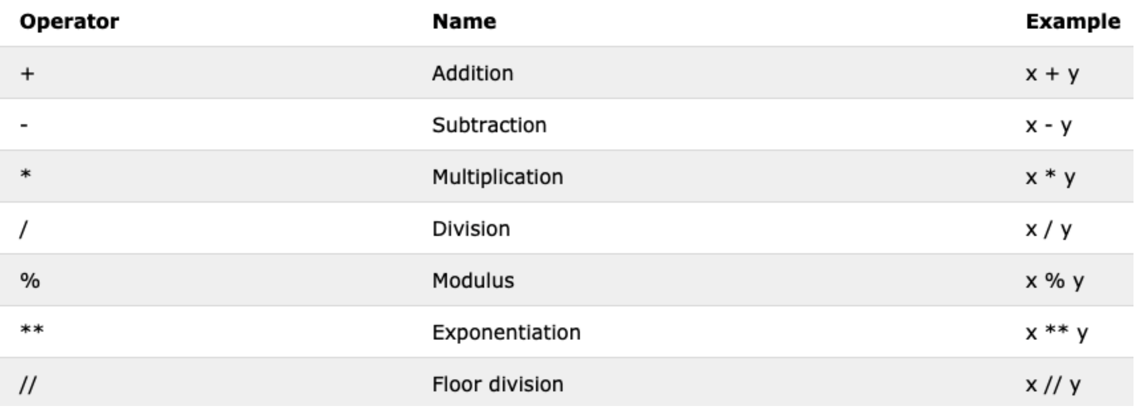
```
x = str(3)     # x will be '3'
y = int(3)     # y will be 3
z = float(3)   # z will be 3.0
```

- Be careful when casting floats to ints. **The float constructor does not round, it throws away the decimal.**

Demo: Printing variables

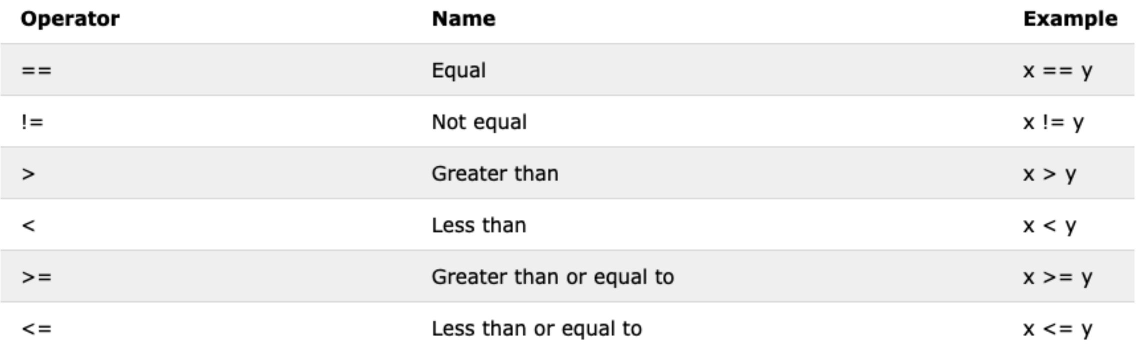
- print statement: `print(’hi’)`
- concatenate strings: `name=‘ulab’; print(‘hi ’ + name)`
- What happens when? `age=5; print(‘age is ’ + age)`. Demo: reading stacktrace. We need to cast!
- String formatting: `‘my name is {} and I am {} years old’.format({’ulab’}, {5})`. You can read up on more advance formatting operations

Mathematical operations:



Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

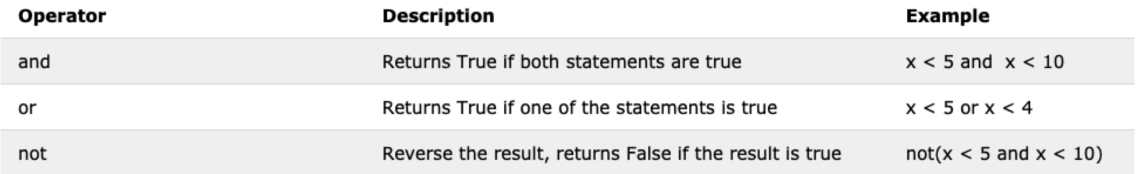
Comparison operations:



Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Explain: we use == rather than = because = is used to assign values.

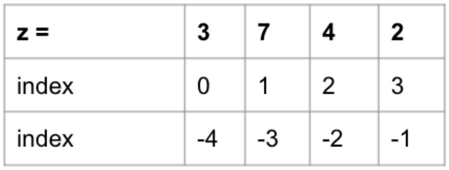
Logic operations:



Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

5 Lists

- Lists are ordered and zero indexed



z =	3	7	4	2
index	0	1	2	3
index	-4	-3	-2	-1

- Reminder: elements do not have to be the same type: `z = ['ulab', 5]`
- Access element: `name = z[0]`
- Change element: `z[1] = z[1] + 1`
- Length of a list: `len(z)`
- Create an empty list: `z = []`

More about lists:

- Negative indexing: index of -1 is the last element of the list. Counting down indexes from back to front.
- Slicing: [start : end : jump]. Demo: `z[1:3], z[:], z[1:], z[:3], z[::2], z[-2:-1]`
- List methods: append(), extend(), index(), more but we use these three the most.

6 Conditionals

We use conditionals when we want a block of code to run IF some CONDITION is true.

- Demo: single if block
- Demo: if else block
- Demo: if elif else block
- Demo: if elif elif ... elif else block
- Notice: the program exits the loop IF an `if` or `elif` block is executed. Subsequent `elif` or `else` blocks (if there are any) are ignored.

7 Next Week

- Loops, functions, more Python!

8 Homework

- For the most part HWs from now on will be in Jupyter notebook. Demo: saving notebook as PDF.
- Practice with Jupyter, variables, math, lists, and conditionals

A (Optional) String Formatting

Formatting text: `print(’pi is ’.format(math.pi))`

A useful list of options for formatting numbers:

Number	Format	Output	Description
3.1415926	{:.2f}	3.14	Format float 2 decimal places
3.1415926	{:+.2f}	+3.14	Format float 2 decimal places with sign
-1	{:+.2f}	-1.00	Format float 2 decimal places with sign
2.71828	{:.0f}	3	Format float with no decimal places
5	{:0>2d}	05	Pad number with zeros (left padding, width 2)
5	{:x<4d}	5xxx	Pad number with x's (right padding, width 4)
10	{:x<4d}	10xx	Pad number with x's (right padding, width 4)
1000000	{:,}	1,000,000	Number format with comma separator
0.25	{:.2%}	25.00%	Format percentage
1000000000	{:.2e}	1.00e+09	Exponent notation
13	{:10d}	13	Right aligned (default, width 10)
13	{:<10d}	13	Left aligned (width 10)
13	{:ˆ10d}	13	Center aligned (width 10)

We can substitute multiple variables:

```
s1 = "cats"
s2 = "dogs"
s3 = " it’s raining {} and {} ".format(s1, s2)

>>> s3
"it’s raining cats and dogs"
```

Using numbered parameters:

```
s = "{0}: Oh {1}, {1}! Wherefore art thou {1}?".format("Juliet", "Romeo")

>>> s
’Juliet: Oh Romeo, Romeo! Wherefore art thou Romeo?’
```

F-Strings:

```
book = "Lord of the Rings"
author = "J.R.R. Tolkien"
s = f"The {book} was written by {author}"

>>> s
’The Lord of the Rings was written by J.R.R. Tolkien’
```