LING 388: Computers and Language

Lecture 5

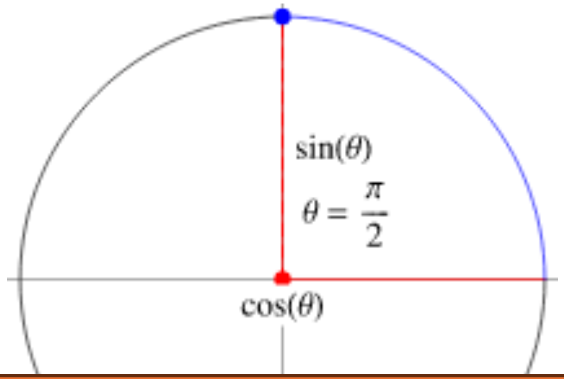
Today's Topic

- Python
  - numbers
  - strings
  - **coercion**: *converting between different types of numbers and strings*
- Homework 4

Python: Numbers

- At the interpreter:

```
$ python3
Python 3.9.12 (main, Jun  1 2022, 06:34:44)
[Clang 12.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> 4+5
9
>>> math.pi
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'math' is not defined
>>> import math
>>> math.pi
3.141592653589793
>>> math.sin(math.pi/2)
1.0
>>>
```



64-bit double precision, with an approximate absolute normalized range of 0 and 10^{-308} to 10^{308} and with a precision of about 16 decimal digits

3.1415 92653 58979 3

π ≈ 3.14159 26535 89793 23846 26433

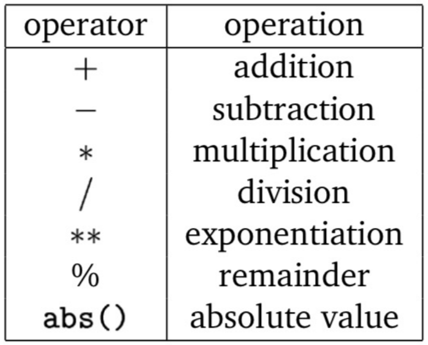
Python: Numbers

type: built-in function

```
>>> type(2*3-1)
<class 'int'>
>>> type(math.pi)
<class 'float'>
>>> import sys
>>> sys.maxsize
9223372036854775807
>>> type(sys.maxsize)
<class 'int'>
>>> type(sys.maxsize+1)
<class 'int'>
>>> sys.int_info
sys.int_info(bits_per_digit=30, sizeof_digit=4)
```

```
math.log(sys.maxsize)/math.log(2)
63.0 (bits)
```

arithmetic operators:

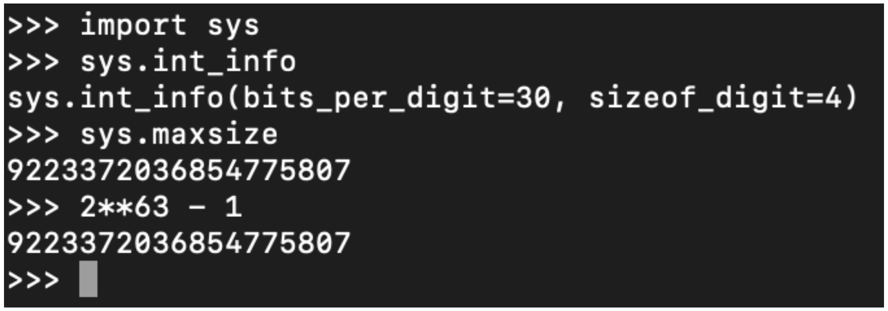


operator	operation
+	addition
−	subtraction
*	multiplication
/	division
**	exponentiation
%	remainder
abs()	absolute value

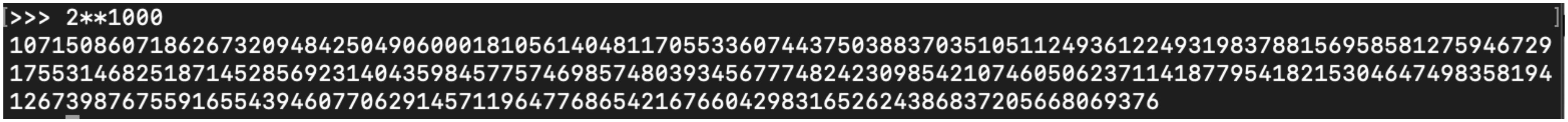
Table 3.1: Python built-in numeric operations.

Python integers

- Python 3: int can go to any size (*limited by available memory*):



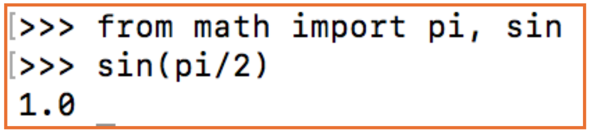
```
>>> import sys
>>> sys.int_info
sys.int_info(bits_per_digit=30, sizeof_digit=4)
>>> sys.maxsize
9223372036854775807
>>> 2**63 - 1
9223372036854775807
>>> 
```



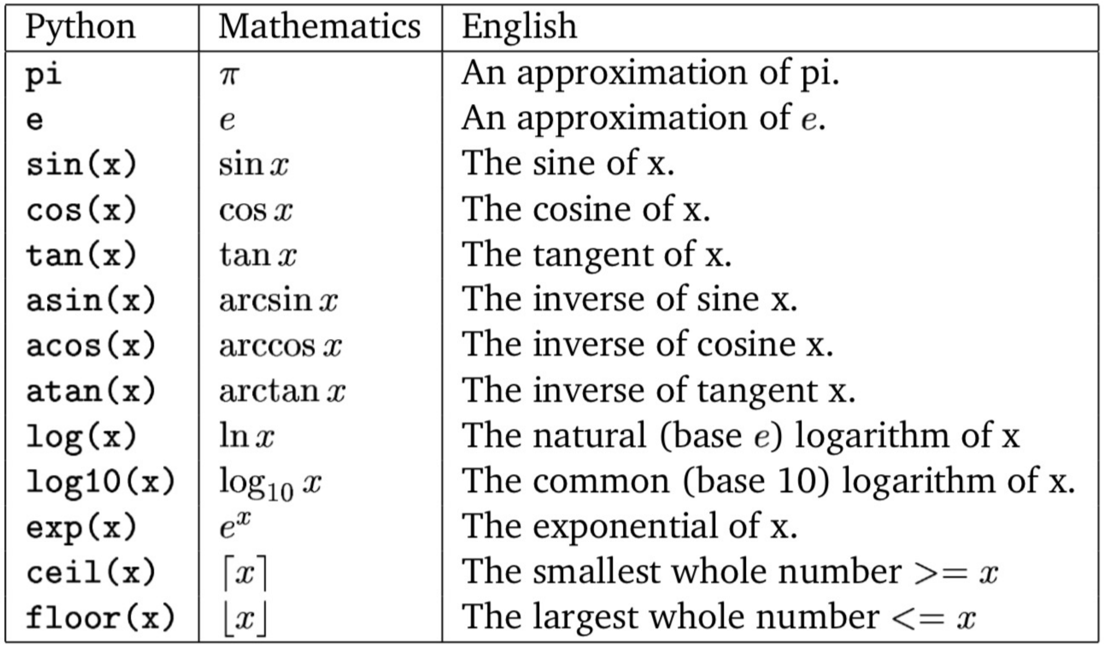
```
[>>> 2**1000
10715086071862673209484250490600018105614048117055336074437503883703510511249361224931983788156958581275946729
17553146825187145285692314043598457757469857480393456777482423098542107460506237114187795418215304647498358194
1267398767559165543946077062914571196477686542167660429831652624386837205668069376
```

Python: Numbers

```
import math
math.pi
```



```
>>> from math import pi, sin
>>> sin(pi/2)
1.0
```



Python	Mathematics	English
`pi`	π	An approximation of pi.
`e`	e	An approximation of e.
`sin(x)`	$\sin x$	The sine of x.
`cos(x)`	$\cos x$	The cosine of x.
`tan(x)`	$\tan x$	The tangent of x.
`asin(x)`	$\arcsin x$	The inverse of sine x.
`acos(x)`	$\arccos x$	The inverse of cosine x.
`atan(x)`	$\arctan x$	The inverse of tangent x.
`log(x)`	$\ln x$	The natural (base e) logarithm of x
`log10(x)`	$\log_{10} x$	The common (base 10) logarithm of x.
`exp(x)`	e^x	The exponential of x.
`ceil(x)`	$\lceil x \rceil$	The smallest whole number $>= x$
`floor(x)`	$\lfloor x \rfloor$	The largest whole number $<= x$

Table 3.2: Some math library functions.

Python: complex numbers

- Example:

  sqrt is the square root function, e.g. `sqrt(4)=2, sqrt(9)=3` etc.

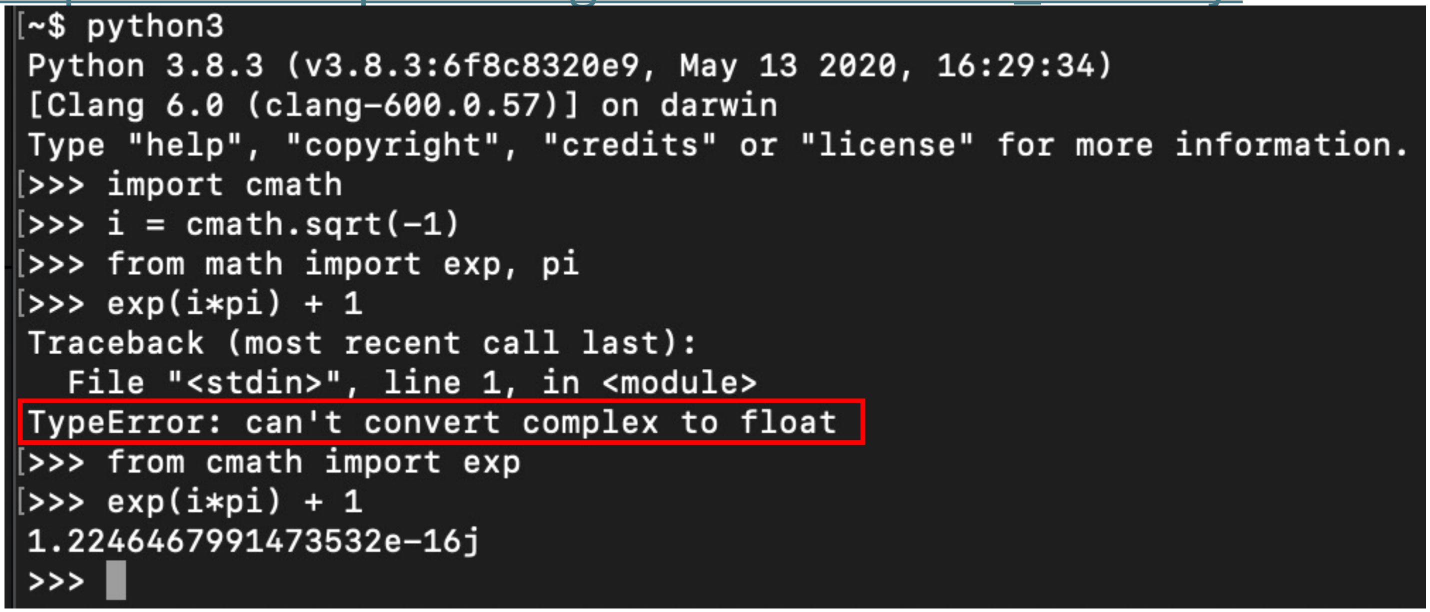
  ```
  >>> math.sqrt(-1)
  Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
  ValueError: math domain error
  ```

- Complex number library:
  - https://docs.python.org/3/library/cmath.html
  - *i* is *j* in Python

  ```
  >>> import cmath
  >>> cmath.sqrt(-1)
  1j
  >>> i = cmath.sqrt(-1)
  >>> i*i
  (-1+0j)
  ```

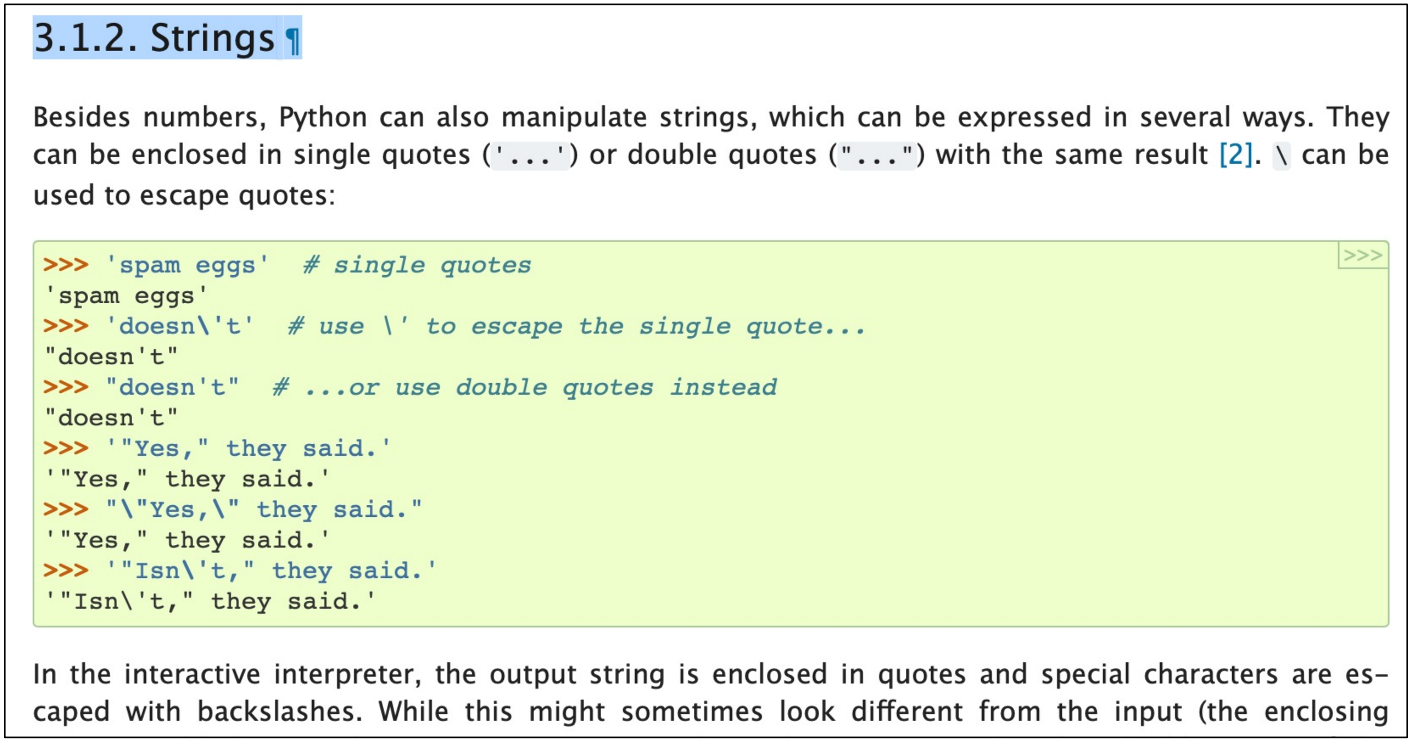
Python: complex numbers

- Euler’s Identity: $$e^{i\pi} + 1 = 0$$
- https://en.wikipedia.org/wiki/Euler%27s_identity



```
[~$ python3
 Python 3.8.3 (v3.8.3:6f8c8320e9, May 13 2020, 16:29:34)
 [Clang 6.0 (clang-600.0.57)] on darwin
 Type "help", "copyright", "credits" or "license" for more information.
[>>> import cmath
[>>> i = cmath.sqrt(-1)
[>>> from math import exp, pi
[>>> exp(i*pi) + 1
 Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
 TypeError: can't convert complex to float
[>>> from cmath import exp
[>>> exp(i*pi) + 1
 1.2246467991473532e-16j
 >>> 
```

Python: Strings



3.1.2. Strings ¶

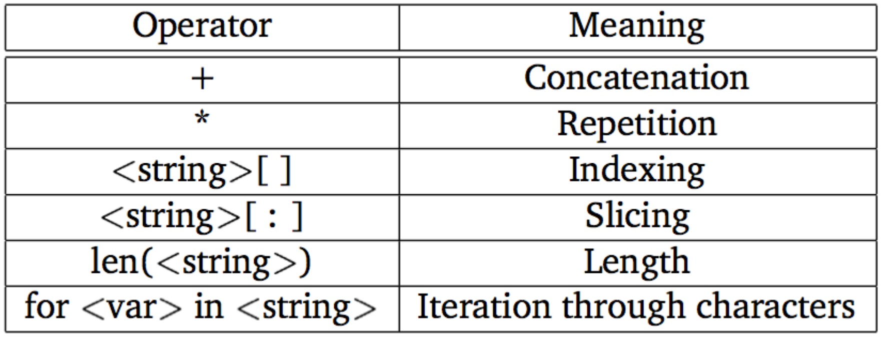
Besides numbers, Python can also manipulate strings, which can be expressed in several ways. They can be enclosed in single quotes (`'...'`) or double quotes (`"..."`) with the same result [2]. `\` can be used to escape quotes:

```
>>> 'spam eggs'  # single quotes
'spam eggs'
>>> 'doesn\'t'  # use \' to escape the single quote...
"doesn't"
>>> "doesn't"  # ...or use double quotes instead
"doesn't"
>>> '"Yes," they said.'
'"Yes," they said.'
>>> "\"Yes,\" they said."
'"Yes," they said.'
>>> '"Isn\'t," they said.'
'"Isn\'t," they said.'
```

In the interactive interpreter, the output string is enclosed in quotes and special characters are escaped with backslashes. While this might sometimes look different from the input (the enclosing

String indexing and slicing

- String is like an array of characters (strings):
  - **`str[i]`** index i (from 0 to len(str)-1)
  - **`str[-i]`** index i from the end (1 = last)
  - **`str[i:j]`** slice from index i until index j-1
  - **`str[:j]`** slice from index 0 until index j-1
  - **`str[i:]`** slice from index i until end of the string



Operator	Meaning
+	Concatenation
*	Repetition
<string>[ ]	Indexing
<string>[:]	Slicing
len(<string>)	Length
for <var> in <string>	Iteration through characters

Table 4.1: Python string operations.

Python numbers and strings

- Explicit type coercion functions:
  1. `float()`
  2. `int()`
  3. `long()`
     - `64 bit integer:` *`not in Python 3`*
  4. `complex(`*`real`*`,`*`imaginary`*`)`
     - a *complex number out of two floating point numbers*
  5. `complex(`*`string`*`)`
     - e.g. `complex('0+1j')`
  6. `str(`*`number`*`)`

Homework 4

- Look up the Python function `ord()`
- See also reference slide (*next slide*) presented before for other relevant functions

1. Compute the hexadecimal representation of the UTF-8 encoding of your first name. Put the hex characters together. Show your work.
   - Example: John is `4a6f686e` in hex
2. What is the decimal and binary representation of your answer to Q1?
   - Example: `0x4a6f686e` is 1248815214 is 0b1001010011011110110100001101110

Homework 4

3. Let's flip the bits in your name. We can use the XOR operator (^) to do this as follows. 0 ^ 1 = 1, 1 ^ 1 = 0. Convert the number back to hexadecimal.
   - Example:

```
0x4a6f686e ^ 0xffffffff is 3046152081
hex(0x4a6f686e ^ 0xffffffff) is '0xb5909791'
```

(**note**: *8* `f`*'s in* `0xffffffff` *because we match the number of hex digits in* `4a6f686e`)

or John has 4 characters, each character has 2 hex digits (1 byte), total is 8.

Homework 4

4. Look up each pair of hex digits (i.e. byte) using function `chr()` to find your flipped name.

See also https://www.charset.org/utf-8

- Example:

`b5909791` is `b5 90 97 91`. And chr(0xb5) is 'µ'.

`John` flipped is `µ\x90\x97\x91` printable part is just µ

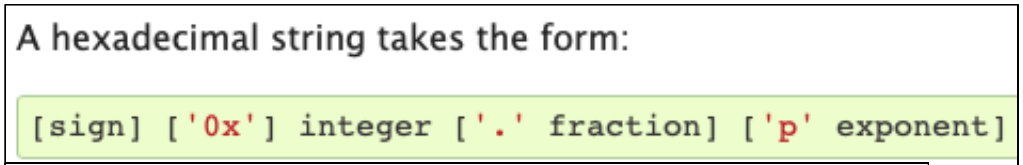
`\x90` etc. are not utf-8 (i.e. *not a valid character*)

But `JOHN` flipped is more interesting: µ°·±

(*so I recommend starting with your name in uppercase*)

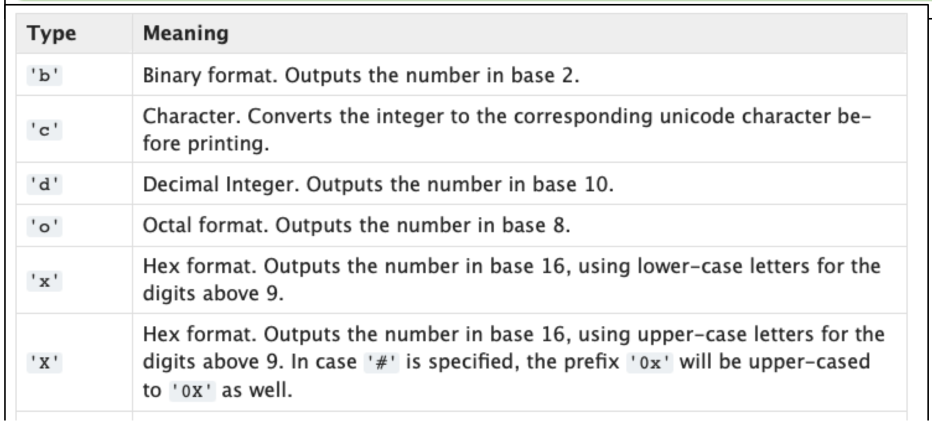
Python: number systems

```
 $ python
Python 3.9.12 (main, Jun  1 2022, 06:34:44)
[Clang 12.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> bin(91)
'0b1011011'
>>> hex(91)
'0x5b'
>>> 0b1011011
91
>>> oct(91)
'0o133'
>>>
```



A hexadecimal string takes the form:

```
[sign] ['0x'] integer ['.' fraction] ['p' exponent]
```



Type	Meaning
`'b'`	Binary format. Outputs the number in base 2.
`'c'`	Character. Converts the integer to the corresponding unicode character before printing.
`'d'`	Decimal Integer. Outputs the number in base 10.
`'o'`	Octal format. Outputs the number in base 8.
`'x'`	Hex format. Outputs the number in base 16, using lower-case letters for the digits above 9.
`'X'`	Hex format. Outputs the number in base 16, using upper-case letters for the digits above 9. In case `'#'` is specified, the prefix `'0x'` will be upper-cased to `'0X'` as well.

Homework 4

- Submit to sandiway@arizona.edu
- SUBJECT: 388 Homework 4 *YOUR NAME*
- One PDF file only
  - include Python terminal screenshots in your answer
- Deadline:
  - midnight Monday
  - we will review the homework on Tuesday