You can access these slides on the course Github:
https://github.com/natrask/ENM1050

ENGR 1050
Intro to Scientific Computation

Lecture 02 – Lists, file I/O, and plotting

Prof. Nat Trask
Mechanical Engineering & Applied Mechanics
University of Pennsylvania

Check our progress

So far we've learned …

Printing: `print("something")`

Comments: `# this is a comment`

Variables:

```
my_name = "nat"
my_num = 3.1415
my_bool = True
```

Operators:

```
my_num*4 = 12.566
my_name*2 = "natnat"
```

Functions

A **function** is a sequence of statements that have been given a name

Functions **take** arguments and **return** results

```
print(3)
abs(-10)
```

arguments

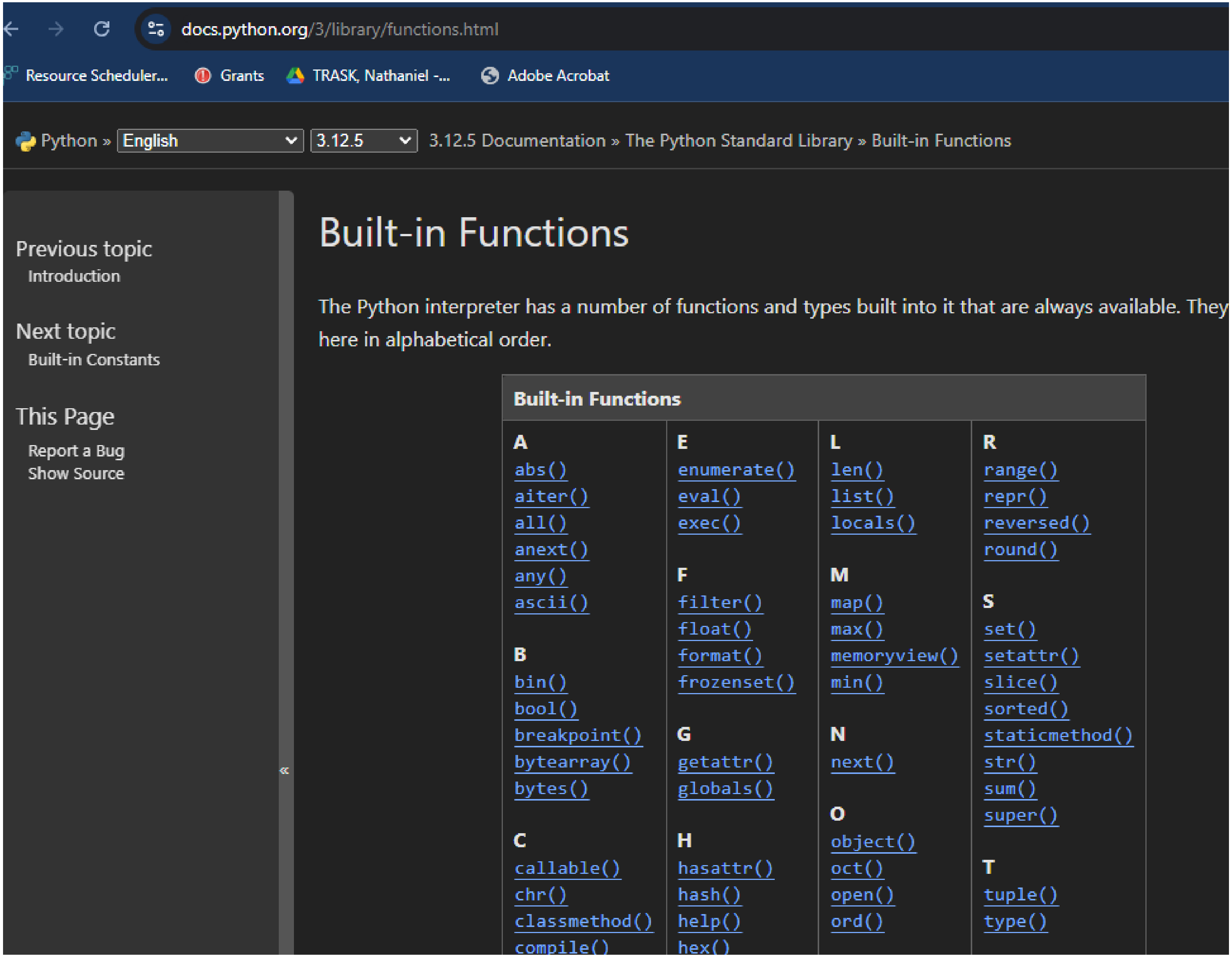
```
3
10
```

result

Python’s Built-In Functions

abs()	format()	list()	round()
all()	getattr()	locals()	set()
any()	globals()	map()	setattr()
ascii()	hasattr()	max()	slice()
bool()	hash()	min()	sorted()
delattr()	help()	next()	str()
dict()	hex()	object()	sum()
dir()	id()	oct()	super()
divmod()	input()	open()	tuple()
enumerate()	int()	pow()	type()
eval()	isinstance()	print()	vars()
exec()	issubclass()	range()	zip()
filter()	iter()	repr()	__import__()
float()	len()	reversed()	

Check the docs!



docs.python.org/3/library/functions.html

Resource Scheduler... Grants TRASK, Nathaniel -... Adobe Acrobat

Python » English 3.12.5 3.12.5 Documentation » The Python Standard Library » Built-in Functions

Previous topic
Introduction

Next topic
Built-in Constants

This Page
Report a Bug
Show Source

Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They
here in alphabetical order.

Built-in Functions

A
abs()
aiter()
all()
anext()
any()
ascii()

B
bin()
bool()
breakpoint()
bytearray()
bytes()

C
callable()
chr()
classmethod()
compile()

E
enumerate()
eval()
exec()

F
filter()
float()
format()
frozenset()

G
getattr()
globals()

H
hasattr()
hash()
help()
hex()

L
len()
list()
locals()

M
map()
max()
memoryview()
min()

N
next()

O
object()
oct()
open()
ord()

R
range()
repr()
reversed()
round()

S
set()
setattr()
slice()
sorted()
staticmethod()
str()
sum()
super()

T
tuple()
type()

https://docs.python.org/3/library/functions.html

Debugging

Programming is complicated! We all make mistakes.

Errors are called **bugs**.

Finding and **removing bugs** is called **debugging**.

Code style

To limit errors and make debugging easy, your code should always be

- readable
- consistent
- commented and/or documented

Code that is easy to read and understand is far more likely to be correct!

Code approach

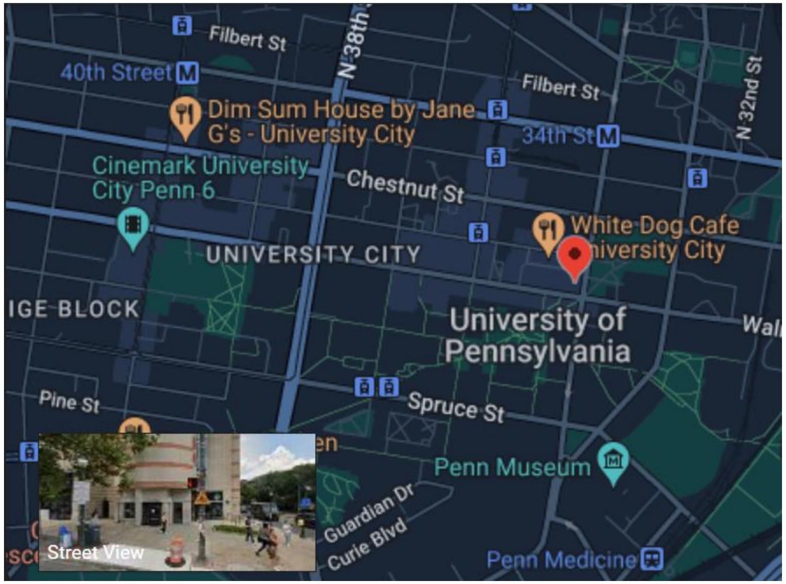
- Start with a plan
- Outline your code using comments (pseudocode)
- Fill in chunks of the code and check as you go

Office Hours

We'll be holding the following OH

- Dr. Trask hours
    - Nat's office: PICS 532
    - Tu 11-12
    - Th 11-12
- TA hours
    - PICS conference room 517
    - Option 1: Wed 12-130
    - Option 2: Fri 11-1
- To listen in on Zoom
    - **Wed OH**
      Meeting ID: 746 8391 6021
      Passcode: 1YN4nn

All OH will be held in the Penn Institute for Computational Science (PICS)



- **3401 Walnut (1 door west of the starbucks), 5th floor**
- **Swipe access coming soon**

Introducing
Our first simulator

(Preview of what we'll build toward in the next couple weeks)

Introducing
Our first simulator

A **differential equation** is an equation relating a function to its derivatives and describes the evolution of a system over time

$$F = ma \longrightarrow m\ddot{x} = F(x)$$

High school physics | **Differential equation version**

In engineering, we will derive models that give lots of complicated equations relating derivatives

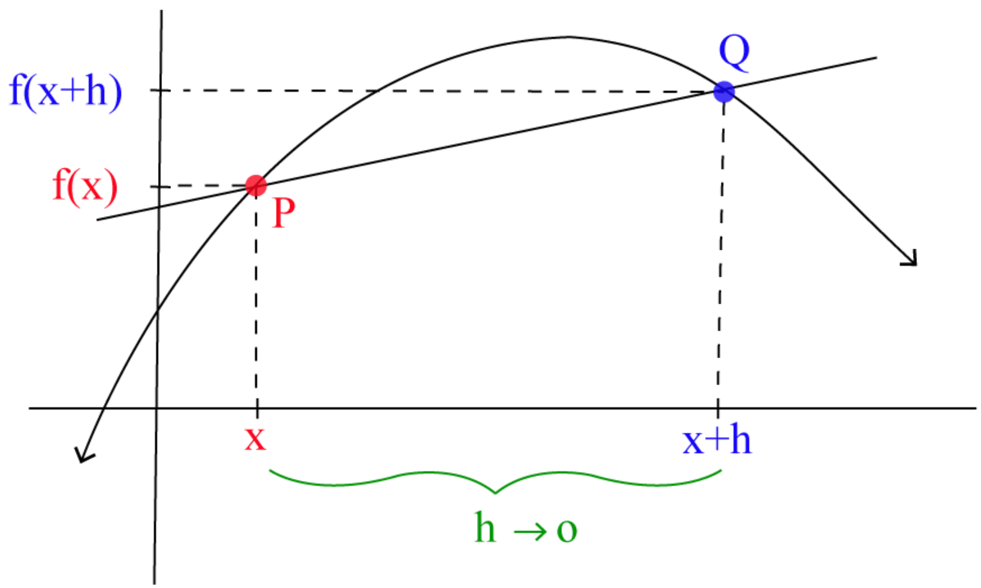
$$G(x, \dot{x}, \ddot{x}, \dddot{x}, \ldots) = 0$$

Introducing
Our first simulator

How do we get this onto a computer?

In calc you'll learn the definition of a derivative

$$\dot{x}(t) = \lim_{h \to 0} \frac{x(t+h) - x(t)}{h}$$



The explicit Euler approximation

Given a differential equation:

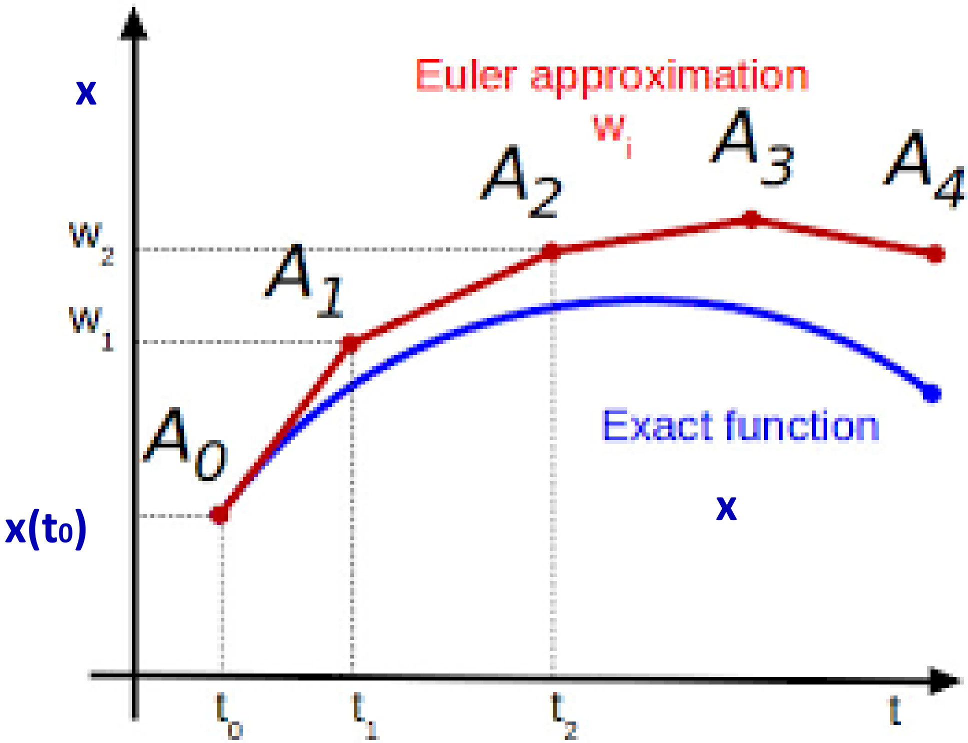
$$\dot{x} = f(x)$$

Approximate the derivative:

$$\dot{x}(t) = \lim_{h \to 0} \frac{x(t+h) - x(t)}{h}$$

Substitute it in at time n

$$\frac{x(t_{n+1}) - x(t_n)}{t_{n+1} - t_n} = f(x(t_n))$$



Reorganize to get an update rule

$$x(t_{n+1}) = x(t_n) + (t_{n+1} - t_n) f(x(t_n))$$

Lists, loops, ifs, plots: ingredients of a simulator

Store solution as a list of points

```
xsol= [x0,x1,x2,x3,x4]
```

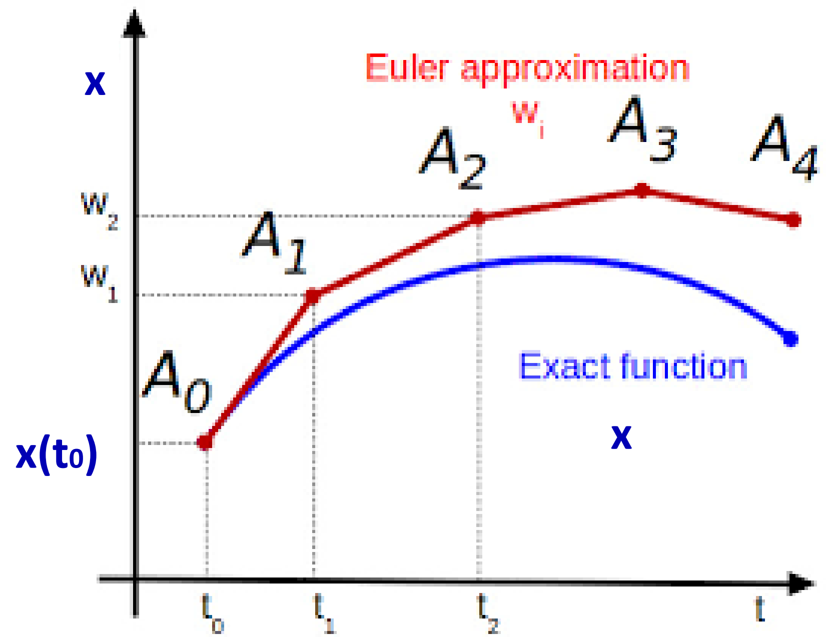
Use for loops to update answer at each step

$$x(t_{n+1}) = x(t_n) + (t_{n+1} - t_n) f(x(t_n))$$

Use if statements check if we're at final time

```
if (tn < t4):
```

To visualize prediction, plot solution



These are the basic building blocks for writing programs, and what we'll focus on in the next few lectures

Today – lists and plots

Last Time: Variable Types

`str` (string)

- Ex: `"ENGR 1050"`, `'a’`, `“Nat"`

`int` (integer)

- Ex: `2`, `73`, `-122`

`float` (floating point number)

- Ex: `3.14`, `-18.0`

`bool` (boolean)

- can be `True` or `False` only

Variables

- Reference them by the name

```
a_number = 23
a_number + 2
```

- Variables and strings are not the same thing

```
a_string = "hello“
print(a_string)
```

~~`print(hello)`~~

Another type of variable: Lists

A **list** is a sequence of values

- each **element** (value) is identified by an **index** (starting from 0)

Create a list by naming it and assigning values in (square) brackets, with values separated by commas

```
list_of_ints = [10, 20, 30, 50]
print(list_of_ints[0]) % outputs 10
```

Another type of variable: Lists

A **list** is a sequence of values

- each **element** (value) is identified by an **index** (starting from 0)
- the elements of the list can be of any type

```
tens = [10, 20, 30, 40]
TAs = [“abdullah", “michelle", “alfredo“]
empty = []
```

- lists can have mixed types and even other lists (nested list)

```
mixed = ["hello", 2.0, 5, [10, 20]]
```

Lists are useful for plotting

Prompt from your math class:

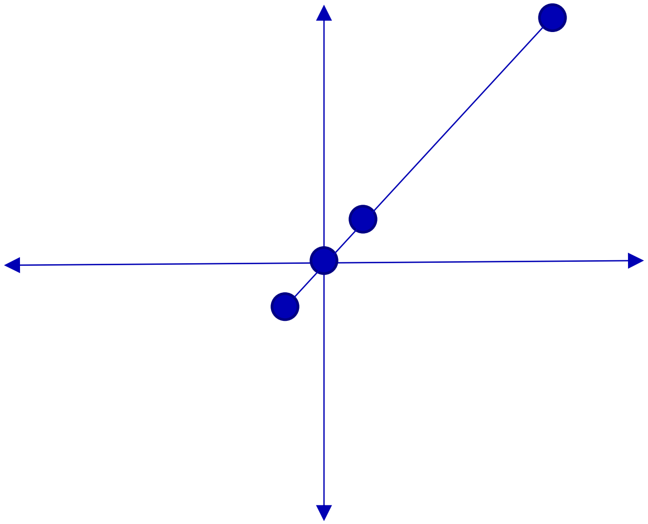
Plot the curve $f(x) = x$

Algorithm:

1. Check some points
   $f(0) = 0, f(1) = 1, f(-1) = -1, f(10) = 10$

2. Plot those points

3. Connect them with a line/curve



Lists are used to store (x,y) pairs to plot

Prompt from your math class:

Plot the curve $f(x) = x$

Algorithm:

1. Check some points
   $f(0) = 0, f(1) = 1, f(-1) = -1, f(10) = 10$

2. Plot those points

3. Connect them with a line/curve

In-Class: 02_Plotting

Do this with a partner. Turn in as a pair on Canvas.

Tips for pair programming:

- Switch off who is typing.
- The person who is not typing should:
  - Make comments or suggest potential solutions
  - Be “devil’s advocate”: what are potential issues with what is being typed
  - Suggest other things to explore

At-Home: posted on Canvas. Do this individually.