Session_1

October 5, 2022

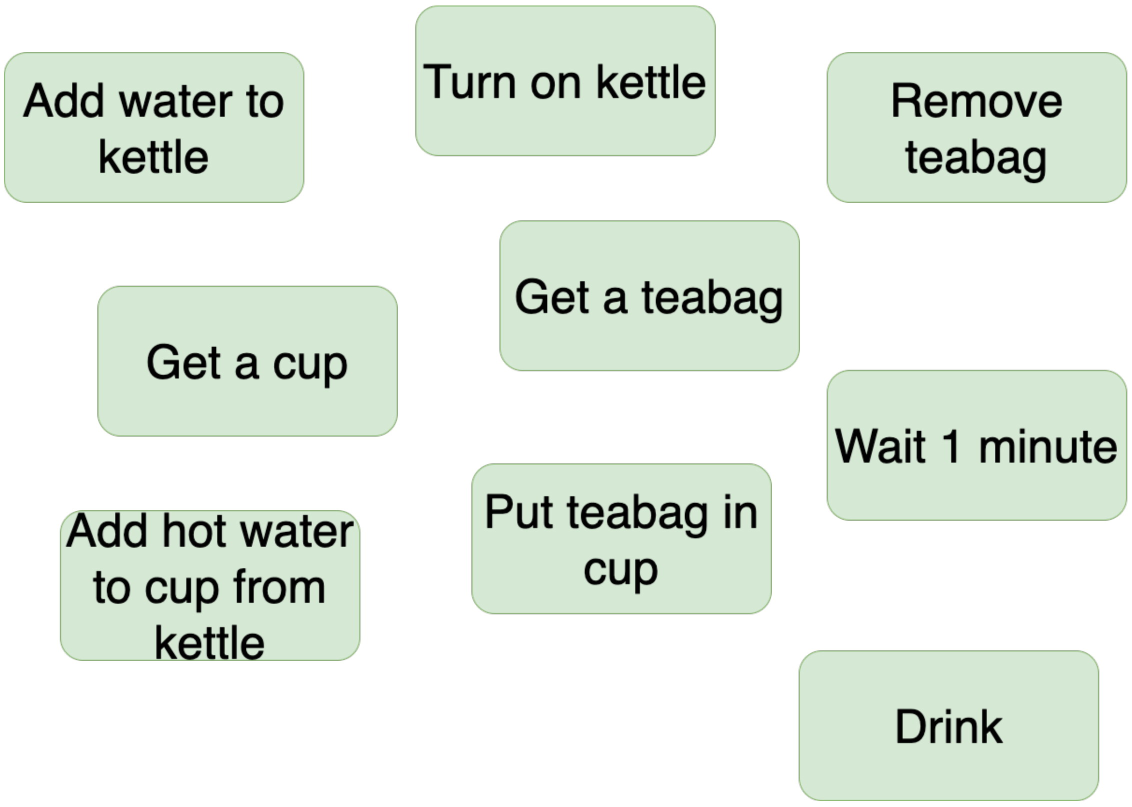
1 Programming introduction

1.1 Why learn to code

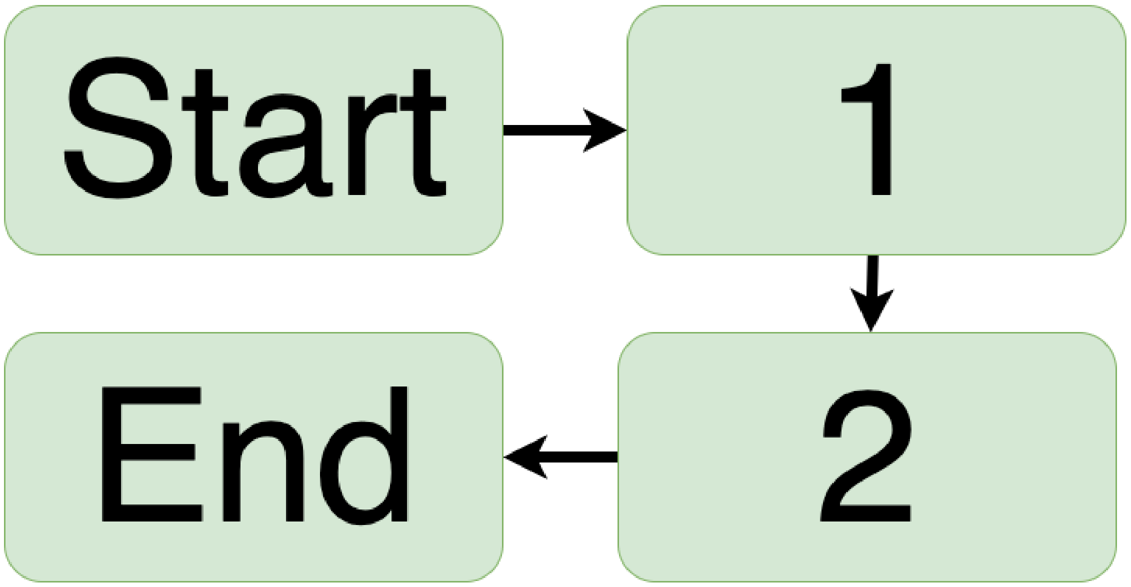
- To solve problems
- Save time
  - Al Sweigart has an amazing book on automating with python
- Consistently
  - if you do the same thing all the time, the same way, why do it?
- Challenge
- Fun!

1.2 How do Computers think? (Let's make some tea)

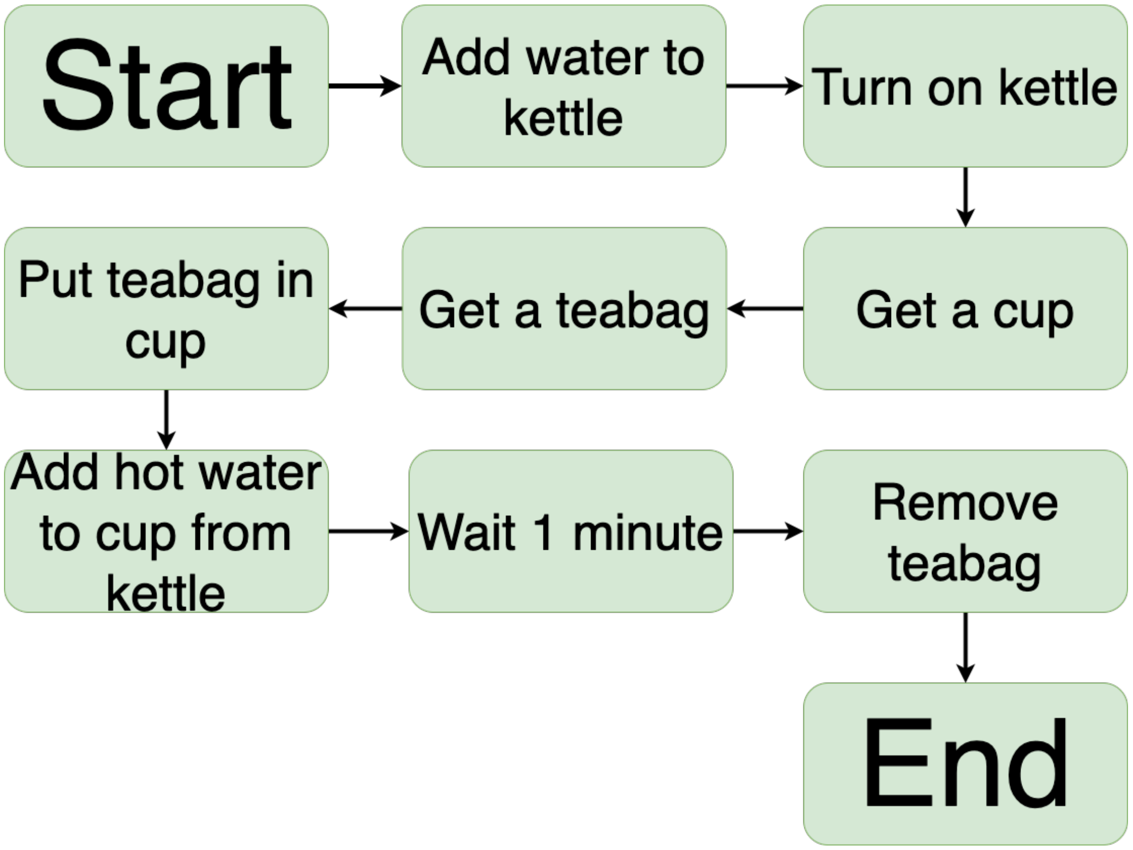
1.2.1 A human process



1.2.2 A computer process



1.2.3 A computer making tea



1.3 Use cases (session 2 preview)

In session 2 we will look at some practical use cases for the programming skills learnt here

1. Making a number guessing game
2. Reading/Writing Excel documents
3. Writing codes and ciphers
4. Automatically get information from websites

2 Introduction to Python

2.1 Reading code

One of the best things about the language Python is that it is so easily readable, at least compared with other contemporary languages.

For example let's interpret the following code snippets and estimate what they do

2.1.1 Example 1

[1]:
```python
x = 10
print(x)
```

```
10
```

2.1.2 Example 2

[2]:
```python
counter = 0
counter = counter + 1
print(counter)
```

```
1
```

2.1.3 Example 3

[3]:
```python
room = ["person1","person2","person3","person4"]
number_of_people = 0

for person in room:
    number_of_people = number_of_people + 1

print(number_of_people)
```

```
4
```

2.2 Basic Operations

[4]:
```
# There are many basic mathematics operations, most common are

# Addition

1+1
```

[4]:
```
2
```

[5]:
```
# Subtraction

1-1-1-1-1
```

[5]:
```
-3
```

[6]:
```
# Multiplication and division

10*10/5
```

[6]:
```
20.0
```

[7]:
```
# Powers

10**2
```

[7]:
```
100
```

[8]:
```
# Note that order of operations, like in Maths, is very important. We can use ()␣
 ↪to be more specific if we want

5*5+5/2
```

[8]:
```
27.5
```

[9]:
```
(5*5+5)/2
```

[9]:
```
15.0
```

2.3 Data types

- All we've looked at so far has been numbers
- But what if we want to use non-numerical data?

For ~99% of the code we write we can largely just think about 4 kinds of data: 1. Numbers - Integers: 1, 10, 54 - Floating point numbers: 0.1, 1/3, 100.001 2. Strings (text) - "need to have quotes around them" - 'single is allowed too' - "They can be mixed to allow the string to 'include' them" 3. Booleans - True - False 4. Collections of data - [] - list - {} - dictonary - () - set

2.4 Variables

```
[10]: i = 10

      i
```

```
[10]: 10
```

```
[11]: i+i
```

```
[11]: 20
```

```
[12]: i = i + i
      i
```

```
[12]: 20
```

```
[13]: # Using variables allows us to make it more clear what we are doing and limit␣
       ↪mistakes

      x = 5
      y = 10
      z = 20

      volume = x*y*z
      volume
```

```
[13]: 1000
```

```
[14]: # Variables don't have to be numbers, or even single things, as we'll find out␣
       ↪shortly.

      hello = "Good morning"
      name = 'Bob'

      hibob = hello + name
      hibob
```

```
[14]: 'Good morningBob'
```

```
[15]: # let's see if we can fix to make it pretty

      hibob = hello + ' ' + name
      hibob
```

```
[15]: 'Good morning Bob'
```

[16]:
```
# But be careful and know what variable types you're using...
# Is this the output you really want?
name + 5
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/f4/m42_v8vj1fjc0wrwtb69xjt80000gr/T/ipykernel_72048/3397664973.py in␣
 ↪<cell line: 3>()
      1 # But be careful and know what variable types you're using...
      2 # Is this the output you really want?
----> 3 name + 5

TypeError: can only concatenate str (not "int") to str
```

[17]:
```
# Or how about this

hibob * 5
```

[17]:
```
'Good morning BobGood morning BobGood morning BobGood morning BobGood morning
Bob'
```

2.5 Functions

[18]:
```
def myFirstFunction(a):
  return a*2

myFirstFunction(5)
```

[18]:
```
10
```

[19]:
```
# Combine with the previous mention of variables and we can do some fun things

res = myFirstFunction(10)
res = myFirstFunction(res)
res
```

[19]:
```
40
```

[20]:
```
# Or some crazy things, that may be quick to code but hard to read and for␣
 ↪others to understand

res = myFirstFunction(myFirstFunction(10))
res
```

[20]:
```
40
```

[21]:
```
# Using the print function for more output than just the last command entered:

def volumeOfCuboid(x,y,z):
    print("X is equal to", x)
    print("Y is equal to", y)
    print("Z is equal to", z)
    print("----")
    volume = x*y*z
    print("Volume is:")
    print(volume)
    return volume

v = volumeOfCuboid(10,20,30)

a = 5
b = 10
c = 2

v = volumeOfCuboid(a,b,c)
```

```
X is equal to 10
Y is equal to 20
Z is equal to 30
----
Volume is:
6000
X is equal to 5
Y is equal to 10
Z is equal to 2
----
Volume is:
100
```

2.6 Lists, dictionaries and loops

[22]:
```
myShoppingList = ['milk', 'bread', 'tofu']
print(myShoppingList)
print('Opps I forgot to add apples')
myShoppingList.append('apples')
print(myShoppingList)
```

```
['milk', 'bread', 'tofu']
Opps I forgot to add apples
['milk', 'bread', 'tofu', 'apples']
```

[23]:
```
myShoppingList = ['milk', 'bread', 'tofu']
```

```
print("the first thing in my list is")
print(myShoppingList[0])
```

```
the first thing in my list is
milk
```

[24]:
```
myShoppingList = ['milk', 'bread', 'tofu']

for i in myShoppingList:
    print(i)
```

```
milk
bread
tofu
```

[25]:
```
Ages = {"Ben" : 10, "Spider-man": 34, "Claire": 24}
print(Ages["Ben"])
```

```
10
```

[26]:
```
# notice what happens here, is it what you expect?
for i in Ages:
  print(i)
```

```
Ben
Spider-man
Claire
```

[27]:
```
Ages = {"Ben" : 10, "Spider-man": 34, "Claire": 24}

print("ahh, we forgot to get Tom's age")
Ages['Tom'] = 17

print(Ages)
```

```
ahh, we forgot to get Tom's age
{'Ben': 10, 'Spider-man': 34, 'Claire': 24, 'Tom': 17}
```

2.7 Conditional statements

[28]:
```
# Sometimes we want to only process data if certain conditions are met...
# Or we might want to check if something is True, this can be done really easily.
 ↪..



5 == 10
```

[28]:
```
False
```

[29]:
```
10==10
```

[29]:
```
True
```

[30]:
```
'carrots' == 'carrotz'
```

[30]:
```
False
```

[31]:
```
'carrots' == 'carrotz'
```

[31]:
```
False
```

[32]:
```
# Read as "10 is NOT equal to 10" which is a False statement.
10 != 10
```

[32]:
```
False
```

[33]:
```
# We can use an 'in' statement in a similar way

'spider' in 'spider-man'
```

[33]:
```
True
```

[34]:
```
# let's build a piece of code which only runs if a condition is met

age = 16

if age < 17:
  print('Youre too young to drive')
if age >= 17:
  print('You can get a license')
```

```
Youre too young to drive
```

[35]:
```
# we can shorten this

if age < 17:
  print('Youre too young to drive')
else:
  print('you can get a license')
```

```
Youre too young to drive
```

[36]:
```
# Let's just combine some of what we've done together

Ages = {"Ben" : 10, "Spider-man": 34, "Claire": 24}
```

```
for person in Ages:
  if person == "Spider-man":
    print('Hey spidey')
  else:
    if Ages[person] < 18:
      print('Hey, who let a kid in here')
      print('Can ' + person + 's parents come and collect them')
    else:
      print("Hello" + " " + person)
```

```
Hey, who let a kid in here
Can Bens parents come and collect them
Hey spidey
Hello Claire
```

2.8 Importing libraries

[37]:
```
import numpy
i = 100
numpy.sqrt(i)
```

[37]:
```
10.0
```

[38]:
```
import numpy as np
i = 100
np.sqrt(i)
```

[38]:
```
10.0
```

[39]:
```
myArray = np.arange(1,10)
print(myArray)
```

```
[1 2 3 4 5 6 7 8 9]
```

[40]:
```
start = 0
end = 50
num = 100

nums = np.linspace(start, end, num)
print(nums)
```

```
[ 0.          0.50505051  1.01010101  1.51515152  2.02020202  2.52525253
  3.03030303  3.53535354  4.04040404  4.54545455  5.05050505  5.55555556
  6.06060606  6.56565657  7.07070707  7.57575758  8.08080808  8.58585859
  9.09090909  9.5959596  10.1010101  10.60606061 11.11111111 11.61616162
 12.12121212 12.62626263 13.13131313 13.63636364 14.14141414 14.64646465
 15.15151515 15.65656566 16.16161616 16.66666667 17.17171717 17.67676768
 18.18181818 18.68686869 19.19191919 19.6969697  20.2020202  20.70707071
 21.21212121 21.71717172 22.22222222 22.72727273 23.23232323 23.73737374
 24.24242424 24.74747475 25.25252525 25.75757576 26.26262626 26.76767677
 27.27272727 27.77777778 28.28282828 28.78787879 29.29292929 29.7979798
 30.3030303  30.80808081 31.31313131 31.81818182 32.32323232 32.82828283
 33.33333333 33.83838384 34.34343434 34.84848485 35.35353535 35.85858586
 36.36363636 36.86868687 37.37373737 37.87878788 38.38383838 38.88888889
 39.39393939 39.8989899  40.4040404  40.90909091 41.41414141 41.91919192
 42.42424242 42.92929293 43.43434343 43.93939394 44.44444444 44.94949495
 45.45454545 45.95959596 46.46464646 46.96969697 47.47474747 47.97979798
 48.48484848 48.98989899 49.49494949 50.        ]
```

[41]:
```
# we can also do some nice rounding with np
decimals=1
np.around(nums, decimals)
```

[41]:
```
array([ 0. ,  0.5,  1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,  4.5,  5.1,
        5.6,  6.1,  6.6,  7.1,  7.6,  8.1,  8.6,  9.1,  9.6, 10.1, 10.6,
       11.1, 11.6, 12.1, 12.6, 13.1, 13.6, 14.1, 14.6, 15.2, 15.7, 16.2,
       16.7, 17.2, 17.7, 18.2, 18.7, 19.2, 19.7, 20.2, 20.7, 21.2, 21.7,
       22.2, 22.7, 23.2, 23.7, 24.2, 24.7, 25.3, 25.8, 26.3, 26.8, 27.3,
       27.8, 28.3, 28.8, 29.3, 29.8, 30.3, 30.8, 31.3, 31.8, 32.3, 32.8,
       33.3, 33.8, 34.3, 34.8, 35.4, 35.9, 36.4, 36.9, 37.4, 37.9, 38.4,
       38.9, 39.4, 39.9, 40.4, 40.9, 41.4, 41.9, 42.4, 42.9, 43.4, 43.9,
       44.4, 44.9, 45.5, 46. , 46.5, 47. , 47.5, 48. , 48.5, 49. , 49.5,
       50. ])
```

[42]:
```
# Numpy arrays can be accessed similar to lists

nums[10]
```

[42]:
```
5.050505050505051
```

[43]:
```
# and you can do some fun matrix-style maths on them
# notice how each element has been multiplied
nums * 5
```

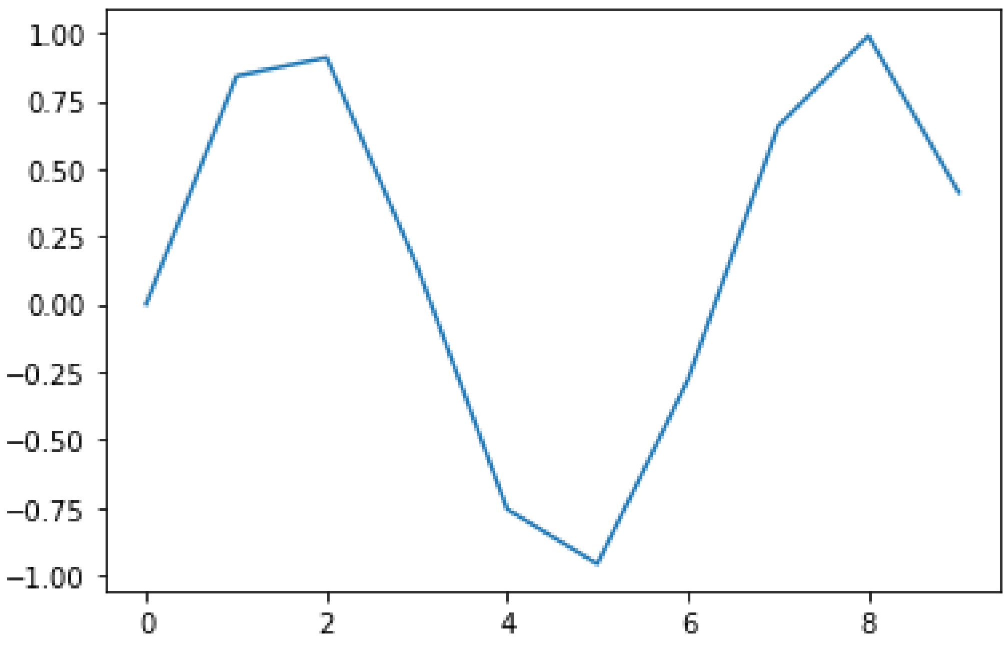
[43]:
```
array([  0.        ,   2.52525253,   5.05050505,   7.57575758,
        10.1010101 ,  12.62626263,  15.15151515,  17.67676768,
        20.2020202 ,  22.72727273,  25.25252525,  27.77777778,
        30.3030303 ,  32.82828283,  35.35353535,  37.87878788,
        40.4040404 ,  42.92929293,  45.45454545,  47.97979798,
        50.50505051,  53.03030303,  55.55555556,  58.08080808,
        60.60606061,  63.13131313,  65.65656566,  68.18181818,
        70.70707071,  73.23232323,  75.75757576,  78.28282828,
        80.80808081,  83.33333333,  85.85858586,  88.38383838,
        90.90909091,  93.43434343,  95.95959596,  98.48484848,
       101.01010101, 103.53535354, 106.06060606, 108.58585859,
       111.11111111, 113.63636364, 116.16161616, 118.68686869,
       121.21212121, 123.73737374, 126.26262626, 128.78787879,
       131.31313131, 133.83838384, 136.36363636, 138.88888889,
       141.41414141, 143.93939394, 146.46464646, 148.98989899,
       151.51515152, 154.04040404, 156.56565657, 159.09090909,
       161.61616162, 164.14141414, 166.66666667, 169.19191919,
       171.71717172, 174.24242424, 176.76767677, 179.29292929,
       181.81818182, 184.34343434, 186.86868687, 189.39393939,
       191.91919192, 194.44444444, 196.96969697, 199.49494949,
       202.02020202, 204.54545455, 207.07070707, 209.5959596 ,
       212.12121212, 214.64646465, 217.17171717, 219.6969697 ,
       222.22222222, 224.74747475, 227.27272727, 229.7979798 ,
       232.32323232, 234.84848485, 237.37373737, 239.8989899 ,
       242.42424242, 244.94949495, 247.47474747, 250.        ])
```

[44]:
```
# Some other common functions with arrays
sums = np.sum(nums)
print("The sum of values is ")
print(sums)

means = np.mean(nums)
print("The mean of values is ")
print(means)

# You can try out others, such as np.min, np.max, np.median, np.std
```

```
The sum of values is
2500.0000000000005
The mean of values is
25.000000000000004
```

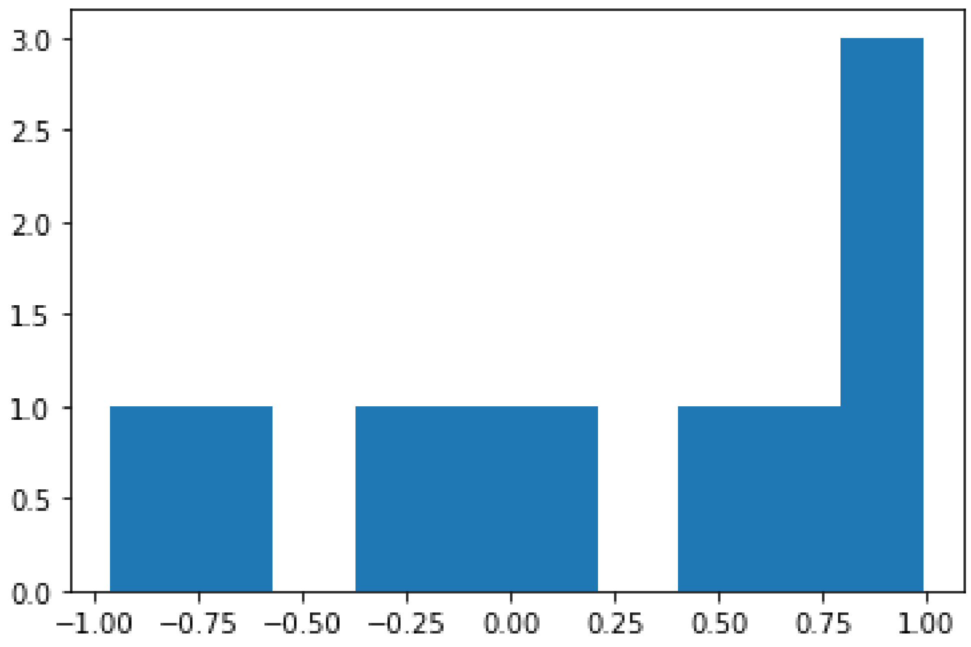
[45]:
```
# Let's do some plotting and really get into the guts of data science

import matplotlib.pyplot as plt
import numpy as np

Xs = np.arange(0,10)
sin_values = [ ]

for v in Xs:
  sin = np.sin(v)
  sin_values.append(sin)

plt.plot(sin_values)
```

[45]:
```
[<matplotlib.lines.Line2D at 0x11695f970>]
```

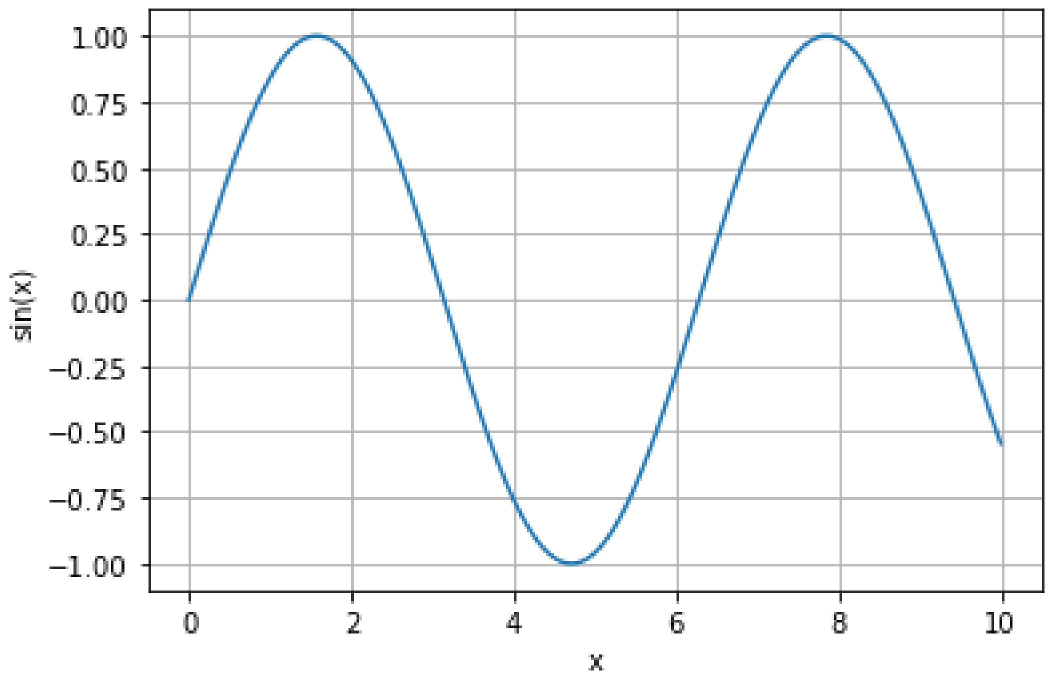


[46]:
```
# We can reuse these same values to do other kinds of plotting... like a␣
 ↪histogram of the values
# Not exciting outputs yet, but good for us to see that we're quickly building␣
 ↪some tools that can help us explore data
plt.hist(sin_values)
```

[46]:
```
(array([1., 1., 0., 1., 1., 1., 0., 1., 1., 3.]),
 array([-0.95892427, -0.76409602, -0.56926777, -0.37443952, -0.17961127,
         0.01521699,  0.21004524,  0.40487349,  0.59970174,  0.79452999,
         0.98935825]),
 <BarContainer object of 10 artists>)
```



[47]:
```
import matplotlib.pyplot as plt
import numpy as np

sin_values = []
Xs = np.linspace(0,10, num=1000)
for x in Xs:
    sin_values.append(np.sin(x))
plt.plot(Xs, sin_values)
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.grid()
```



2.9 Before moving on!

Take some time to look over what we've just done.

Think if you can modify and re-run with other functions. Maybe you can get 'cos' to work rather than 'sin' with a little bit of guess work.

You will probably see some errors if something goes wrong, but that's good! It gives us a chance to learn!

2.10 Some errors and what they mean

[48]:
```
X =
print(X)
```

```
  File "/var/folders/f4/m42_v8vj1fjc0wrwtb69xjt80000gr/T/ipykernel_72048/94210321
↪py", line 1
    X =
        ^
SyntaxError: invalid syntax
```

[49]:
```
X= 10
print(X
  File "/var/folders/f4/m42_v8vj1fjc0wrwtb69xjt80000gr/T/ipykernel_72048/44592098
↪py", line 2
    print(X
          ^
SyntaxError: incomplete input
```

```
[50]: def f(x):
        print(notX)
      f(10)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/var/folders/f4/m42_v8vj1fjc0wrwtb69xjt80000gr/T/ipykernel_72048/1746898150.py in
 ↪<cell line: 3>()
      1 def f(x):
      2   print(notX)
----> 3 f(10)

/var/folders/f4/m42_v8vj1fjc0wrwtb69xjt80000gr/T/ipykernel_72048/1746898150.py in
 ↪f(x)
      1 def f(x):
----> 2   print(notX)
      3 f(10)

NameError: name 'notX' is not defined
```

```
[ ]:
```

```
[ ]:
```