Forward School

Program Code: J620-002-4:2020

Program Name: FRONT-END SOFTWARE DEVELOPMENT

Title : Basic Maths with Python

Name: Chong Mun Chen

IC Number: 960327-07-5097

Date : 10/7/2023

Introduction : Learning basic maths with Python, as well as advanced module on Set.

Conclusion : Understood how to solve problems with values from a standard normal distribution.

P23 - Basic Mathematics with Python

Sets

Python comes with an implementation of the mathematical set. Initially this implementation had to be imported from the standard module set, but with Python 2.6 the types set and frozenset became built-in types. A set is an unordered collection of objects, unlike sequence objects such as lists and tuples, in which each element is indexed. Sets cannot have duplicate members - a given object appears in a set 0 or 1 times. All members of a set have to be hashable, just like dictionary keys. Integers, floating point numbers, tuples, and strings are hashable; dictionaries, lists, and other sets (except frozensets) are not.

Sets in Python at a glance:

In [3]:
```
set1 = set()                    # A new empty set
set1.add("cat")                 # Add a single member
set1.update(["dog", "mouse"])   # Add several members, like list's extend
set1 |= set(["doe", "horse"])   # Add several members 2, like list's extend
if "cat" in set1:               # Membership test
    set1.remove("cat")
#set1.remove("elephant") - throws an error
set1.discard("elephant")        # No error thrown
print (set1)
```

```
{'mouse', 'doe', 'horse', 'dog'}
```

Hints

|= performs an in-place+ operation between pairs of objects. In particular, between:

sets: a union operation dicts: an update operation counters: a union (of multisets) operation numbers: a bitwise OR, binary operation

Reference: https://stackoverflow.com/questions/3929278/what-does-ior-do-in-python (https://stackoverflow.com/questions/3929278/what-does-ior-do-in-python)

Sets

Reference: https://medium.com/better-programming/mathematical-set-operations-in-python-e065aac07413#:~:text=Python's%20set%20is%20an%20unordered,don't%20support%20set%20 (https://medium.com/better-programming/mathematical-set-operations-in-python-e065aac07413#:~:text=Python's%20set%20is%20an%20unordered,don't%20support%20set%20

In [4]:
```
for item in set1:               # Iteration AKA for each element
    print (item)
print ("Item count:", len(set1)) # Length AKA size AKA item count
#1stitem = set1[0]              # Error: no indexing for sets
isempty = len(set1) == 0        # Test for emptiness
#set1 = {"cat", "dog"}          # Initialize set using braces; since Pythor
#set1 = {}                      # No way; this is a dict
set1 = set(["cat", "dog"])      # Initialize set from a list
set2 = set(["dog", "mouse"])
set3 = set1 & set2              # Intersection
set4 = set1 | set2              # Union
set5 = set1 - set3              # Set difference
set6 = set1 ^ set2              # Symmetric difference
issubset = set1 <= set2         # Subset test
issuperset = set1 >= set2       # Superset test
set7 = set1.copy()              # A shallow copy
set7.remove("cat")
print (set7.pop())              # Remove an arbitrary element
set8 = set1.copy()
set8.clear()                    # Clear AKA empty AKA erase
set9 = {x for x in range(10) if x % 2} # Set comprehension
print(set1, set2, set3, set4, set5, set6, set7, set8, set9, issubset, issup
```

```
mouse
doe
horse
dog
Item count: 4
dog
{'dog', 'cat'} {'mouse', 'dog'} {'dog'} {'mouse', 'dog', 'cat'} {'cat'}
{'mouse', 'cat'} set() set() {1, 3, 5, 7, 9} False False
```

Constructing Sets

One way to construct sets is by passing any sequential object to the "set" constructor.

In [5]:
```
set10 = set([0, 1, 2, 3])
print(set10)

set11 = set("obtuse")
print(set11)
```

```
{0, 1, 2, 3}
{'u', 's', 'o', 'e', 't', 'b'}
```

We can also add elements to sets one by one, using the "add" function.

In [8]:
```
s = set([12, 26, 54])
s.add(32)
s
```

Out[8]:
```
{12, 26, 32, 54}
```

Note that since a set does not contain duplicate elements, if we add one of the members of s to s again, the add function will have no effect. This same behavior occurs in the "update" function, which adds a group of elements to a set.

In [9]:
```
s.update([26, 12, 9, 14])
s
```

Out[9]: {9, 12, 14, 26, 32, 54}

Note that you can give any type of sequential structure, or even another set, to the update function, regardless of what structure was used to initialize the set.

The set function also provides a copy constructor. However, remember that the copy constructor will copy the set, but not the individual elements.

In [10]:
```
s2 = s.copy()
s2
```

Out[10]: {9, 12, 14, 26, 32, 54}

In [11]:
```
s2 == s
```

Out[11]: True

Membership Testing

We can check if an object is in the set using the same "in" operator as with sequential data types.

In [12]:
```
print(32 in s)

print(6 in s)

print(6 not in s)
```

```
True
False
True
```

We can also test the membership of entire sets. Given two sets S_1 and S_2, we check if S_1 is a subset or a superset of S_2.

In [13]:
```
print(s.issubset(set([32, 8, 9, 12, 14, -4, 54, 26, 19])))

print(s.issuperset(set([9, 12])))
```

```
True
True
```

Note that "issubset" and "issuperset" can also accept sequential data types as arguments

Reference: https://www.programiz.com/python-programming/methods/set/issuperset (https://www.programiz.com/python-programming/methods/set/issuperset)

Reference: https://www.programiz.com/python-programming/methods/set/issubset (https://www.programiz.com/python-programming/methods/set/issubset)

In [14]:
```
(s.issuperset([32, 9]))
```

Out[14]:
```
True
```

Note that the <= and >= operators also express the issubset and issuperset functions respectively.

In [15]:
```
print(set([4, 5, 7]) <= set([4, 5, 7, 9]))

print(set([9, 12, 15]) >= set([9, 12]))
```

```
True
True
```

Like lists, tuples, and string, we can use the "len" function to find the number of items in a set.

Removing Items

There are three functions which remove individual items from a set, called pop, remove, and discard. The first, pop, simply removes an item from the set. Note that there is no defined behavior as to which element it chooses to remove.

In [16]:
```
s = set([1,2,3,4,5,6])
s.pop()
s
```

Out[16]:
```
{2, 3, 4, 5, 6}
```

We also have the "remove" function to remove a specified element.

In [17]:
```
s.remove(3)
s
```

Out[17]:
```
{2, 4, 5, 6}
```

However, removing a item which isn't in the set causes an error.

In [18]:
```
s.remove(9)
```

```
---------------------------------------------------------------------
--
KeyError                                  Traceback (most recent call las
t)
<ipython-input-18-d6996fc5cb16> in <module>
----> 1 s.remove(9)

KeyError: 9
```

In [19]:
```
s.discard(9)
```

If you wish to avoid this error, use "discard." It has the same functionality as remove, but will simply do nothing if the element isn't in the set

We also have another operation for removing elements from a set, clear, which simply removes all elements from the set.

In [20]:
```
s.clear()
s
```

Out[20]:
```
set()
```

Iteration Over Sets

We can also have a loop move over each of the items in a set. However, since sets are unordered, it is undefined which order the iteration will follow.

In [27]:
```
s = set("blerg")
for n in s:
    print (n)
```

```
r
e
g
l
b
```

Set Operations

Python allows us to perform all the standard mathematical set operations, using members of set. Note that each of these set operations has several forms. One of these forms, `s1.function(s2)` will return another set which is created by "function" applied to S_1 and S_2. The other form, `s1.function_update(s2)`, will change S_1 to be the set created by "function" of S_1 and S_2. Finally, some functions have equivalent special operators. For example, $s1$ & $s2$ is equivalent to `s1.intersection(s2)`.

Intersection

Any element which is in both S_1 and S_2 will appear in their intersection.

In [28]:
```
s1 = set([4, 6, 9])
s2 = set([1, 6, 8])
print(s1.intersection(s2))

print(s1 & s2)

s1.intersection_update(s2)
print(s1)
```

```
{6}
{6}
{6}
```

Union The union is the merger of two sets. Any element in S_1 or S_2 will appear in their union.

In [18]:
```
s1 = set([4, 6, 9])
s2 = set([1, 6, 8])
print(s1.union(s2))

print(s1 | s2)
```

```
{1, 4, 6, 8, 9}
{1, 4, 6, 8, 9}
```

Symmetric Difference

The symmetric difference of two sets is the set of elements which are in one of either set, but not in both.

In [29]:
```
s1 = set([4, 6, 9])
s2 = set([1, 6, 8])
print(s1.symmetric_difference(s2))

set([8, 1, 4, 9])
print(s1 ^ s2)

set([8, 1, 4, 9])
s1.symmetric_difference_update(s2)
print(s1)
```

```
{1, 4, 8, 9}
{1, 4, 8, 9}
{1, 4, 8, 9}
```

Set Difference

Python can also find the set difference of S_1 and S_2, which is the elements that are in S_1 but not in S_2.

In [30]:
```
s1 = set([4, 6, 9])
s2 = set([1, 6, 8])
print(s1.difference(s2))

set([9, 4])
print (s1 - s2)

set([9, 4])
s1.difference_update(s2)
print(s1)
```

```
{9, 4}
{9, 4}
{9, 4}
```

Multiple sets

"union", "intersection", and "difference" can work with multiple input by using the set constructor. For example, using "set.intersection()":

In [31]:
```
s1 = set([3, 6, 7, 9])
s2 = set([6, 7, 9, 10])
s3 = set([7, 9, 10, 11])
set.intersection(s1, s2, s3)
```

Out[31]:
```
{7, 9}
```

frozenset

A frozenset is basically the same as a set, except that it is immutable - once it is created, its members cannot be changed. Since they are immutable, they are also hashable, which means that frozensets can be used as members in other sets and as dictionary keys. frozensets have the same functions as normal sets, except none of the functions that change the contents (update, remove, pop, etc.) are available.

In [5]:
```
fs = frozenset([2, 3, 4])
s1 = set([fs, 4, 5, 6])
s2 = set([fs, 5, 6])
print(s1)
print(s2)

set([4, frozenset([2, 3, 4]), 6, 5])
print(fs.intersection(s1))
print(fs.intersection(s2))

print(fs)
#fs.add(6)
```

```
{frozenset({2, 3, 4}), 4, 5, 6}
{frozenset({2, 3, 4}), 5, 6}
frozenset({4})
frozenset()
frozenset({2, 3, 4})
```

Mathematical Operators

Here are some commonly used mathematical operators

Syntax	Math	Operation Name
a+b	a+b	addition
a-b	a-b	subtraction
a*b	a × b	multiplication
a/b	a ÷ b	division
a//b	[a ÷ b]	floor division (e.g. 5//2=2)
a%b	a mod b	modulo
-a	-a	negation
abs(a)	[a]	absolute value
a**b	a^b	exponent
math.sqrt(a)	$\sqrt{a}$	square root

Note: In order to use the math.sqrt() function, you must explicitly tell Python that you want it to load the math module. To do that, write

```
import math
```

at the top of your file. For other functions made available by this statement, see here (https://docs.python.org/3/library/math.html).

Order of Operations

Python uses the standard order of operations as taught in Algebra and Geometry classes at high school or secondary school. That is, mathematical expressions are evaluated in the following order (memorized by many as PEMDAS), which is also applied to parentheticals.

(Note that operations which share a table row are performed from left to right. That is, a division to the left of a multiplication, with no parentheses between them, is performed before the multiplication simply because it is to the left.)

Name	Syntax	Description	PEMDAS Mnemonic
Parentheses	(...) }	Before operating on anything else, Python must evaluate all parentheticals starting at the innermost level. (This includes functions.)	**P**lease
Exponents	**	As an exponent is simply short multiplication or division, it should be evaluated before them.	**E**xcuse
Multiplication and Division	* / // %	Again, multiplication is rapid addition and must, therefore, happen first.	**M**y Dear
Addition and Subtraction	+ -		**A**unt Sally

Randomness and reproducibility

In Python, we refer to randomness as the ability to generate data, strings, or, more generally, numbers at random.

However, when conducting analysis it is important to consider reproducibility. If we are creating random data, how can we enable reproducible analysis?

We do this by utilizing pseudo-random number generators (PRNGs). PRNGs start with a random number, known as the seed, and then use an algorithm to generate a psuedo-random sequence based on it.

This means that we can replicate the output of a random number generator in python simply by knowing which seed was used.

We can showcase this by using the functions in the python library random.

In [11]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [7]:
```
import random
```

In [15]:
```
random.seed(99)
random.random()
```

Out[15]: 0.40397807494366633

In [37]:
```
random.seed(1234)
random.random()
```

Out[37]: 0.9664535356921388

The random library includes standard distributions that may come in handy

In [38]:
```
# Uniform distribution
random.uniform(25,50)
```

Out[38]: 36.01831497938382

In [39]:
```
mu = 0

sigma = 1

random.normalvariate(mu, sigma)
```

Out[39]: 1.8038006216944658

Lists vs. numpy arrays

Lists can have multiple datatypes. For example one element can be a string and another can be and int and another a float. Lists are defined by using the square brackets: [ ], with elements separated by commas, ','. Ex:

```
my_list = [1, 'Colorado', 4.7, 'rain']
```

Lists are indexed by position. Remember, in Python, the index starts at 0 and ends at length(list)-1. So to retrieve the first element of the list you call:

```
my_list[0]
```

Numpy arrays np.arrays differ from lists is that the contain only 1 datatype. For example all the elements might be ints or strings or floats or objects. It is defined by np.array(object), where the input 'object' can be for example a list or a tuple.
Ex:

```
my_array = np.array([1, 4, 5, 2])
```

or

```
my_array = np.array((1, 4, 5, 2))
```

Lists and numpy arrays differ in their **speed** and **memory efficiency**. An intuitive reason for this is that python lists have to store the value of each element and also the type of each element (since the types can differ). Whereas numpy arrays only need to store the type once because it is the same for all the elements in the array.

You can do calculations with numpy arrays that can't be done on lists.
Ex:

```
my_array/3
```

will return a numpy array, with each of the elements divided by 3. Whereas:

```
my_list/3
```

Will throw an error.

You can appened items to the end of lists and numpy arrays, though they have slightly different commands. It is almost of note that lists can append an item 'in place', but numpy arrays cannot.

```
my_list.append('new item')
np.append(my_array, 5) # new element must be of the same type as all o
ther elements
```

Links to python docs:
Lists (https://docs.python.org/3/tutorial/datastructures.html), arrays

In [65]:
```
my_list = [1, 2, 3]
my_array = np.array([1, 2, 3])
```

In [66]:
```
# Both indexed by position
my_list[0]
```

Out[66]:
```
1
```

In [67]:
```
my_array[0]
```

Out[67]:
```
1
```

In [68]:
```
my_array/3
```

Out[68]:
```
array([0.33333333, 0.66666667, 1.        ])
```

In [69]:
```
my_list/3
```

```
---------------------------------------------------------------------
--
TypeError                                 Traceback (most recent call las
t)
<ipython-input-69-344e5631acd2> in <module>
----> 1 my_list/3

TypeError: unsupported operand type(s) for /: 'list' and 'int'
```

In [70]:
```
my_list.append(5) # inplace
my_list
```

Out[70]:
```
[1, 2, 3, 5]
```

In [73]:
```
my_array = np.append(my_array, 5) # cannot do inplace because not always co
my_array
```

Out[73]:
```
array([1, 2, 3, 5, 5])
```

In [74]:
```
lst = [1, 7, 3, 5]
def get_element(lst):
  new_lst = []
  for i in lst:
    new_lst.append(i**2)
    return lst[1]
get_element(lst)
```

Out[74]:
```
7
```

Exercise: Sets

1. Join these two sets: `set1 = {"a", "b" , "c"}` and `set2 = {1, 2, 3}`
2. Check if "apple" is present in `fruits = {"apple", "banana", "cherry"}`
3. Insert set 2 into set 1. `set1 = {"a", "b" , "c"}` `set2 = {1, 2, 3}`
4. Show the values that appear in both Set A and Set B. `A = {1, 2, 3, 4, 5}` `B = {4, 5, 6, 7, 8}`
5. Show all the values in A and B. `A = {2, 4, 6}` `B = {1, 3, 5}`
6. Show only values that appear in A and not B. `A = {1, 2, 3, 4, 5}` `B = {4, 5, 6, 7, 8}`
7. Show values of A and B without the intersection. `A = {10, 11, 12, 13, 14, 15}` `B = {14, 15, 16, 17, 18}`
8. Extract and print out all the values in Days.
   `Days="Mon","Tue","Wed","Thu","Fri","Sat","Sun"`
9. Compare whether DaysA is a subset of DaysB. `DaysA = "Mon","Tue","Wed"` `DaysB = "Mon","Tue","Wed","Thu","Fri","Sat","Sun"]`
10. Find out whether DaysB is a superset of DaysA.

In [50]:
```
## Q1
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set1.union(set2)
print(set1.union(set2))
```

```
{1, 'a', 2, 3, 'b', 'c'}
```

In [43]:
```
## Q2
fruits = {"apple", "banana", "cherry"}
fruits.issuperset(['apple'])
```

Out[43]: True

In [62]:
```
## Q3
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set1.update(set2)
print(set1)
```

```
{1, 'a', 2, 3, 'b', 'c'}
```

In [55]:
```
## Q4
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
print(A & B)
```

```
{4, 5}
```

In [56]:
```
## Q5
A = {2, 4, 6}
B = {1, 3, 5}
print(A | B)
```

```
{1, 2, 3, 4, 5, 6}
```

In [59]:
```
## Q6
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
print(A.difference(B))
```

```
{1, 2, 3}
```

In [63]:
```
## Q7
A = {10, 11, 12, 13, 14, 15}
B = {14, 15, 16, 17, 18}
print(A.symmetric_difference(B))
```

```
{10, 11, 12, 13, 16, 17, 18}
```

In [64]:
```
## Q8
Days = "Mon","Tue","Wed","Thu","Fri","Sat","Sun"
print(Days)
```

```
('Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun')
```

In [70]:
```
## Q9
DaysA = "Mon","Tue","Wed"
DaysB = "Mon","Tue","Wed","Thu","Fri","Sat","Sun"
DaysA = set(DaysA)
DaysB = set(DaysB)
print(DaysA.issubset(DaysB))
```

```
True
```

In [71]:
```
##Q10
DaysA = "Mon","Tue","Wed"
DaysB = "Mon","Tue","Wed","Thu","Fri","Sat","Sun"
DaysA = set(DaysA)
DaysB = set(DaysB)
print(DaysB.issuperset(DaysA))
```

```
True
```

Exercise: Maths

1. $c = \frac{3+a}{1-n}$, where a=1 and n=10, solve c
2. Find the factorial of 12, $12!$
3. $CI = mean \pm 1.96 * \sqrt{\frac{p(1-p)}{n}}$, where mean=45.6, p = 0.54 and n= 1118. Solve CI.
4. Pythagoras Theorem: $c = \sqrt{a^2 + b^2}$ where a = 3 and b=4, solve C
5. Generate 1000 random normal distribution numbers with mean=50, sd=1 and plot a histogram. Use `seed=1234`
6. Generate 100 random normal distribution numbers with mean=0, sd=1 and plot a qqplot. Use `seed=1234` . Use the `statsmodel.api` library. Interpret the plot.
7. Generate five trials of 10 observations of a uniform random variable on `[0,1)` . Use `seed=1234` . Plot a boxplot. Interpret the plot.
8. Calculate the standard deviation for A. `A = [11,15,17,18,19]` . Standard Deviation, $std = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2}$
9. Solve the following equations using `Sympy library` :
$$2x^2 + y + z = 1$$
$$x + 2y + z = c_1$$
$$-2x + y = -z$$
10. Generate 100 equal distance numbers from the range of 0 to 1 as store into A. Random sampling 50 numbers from A and plot the distribution. Use `seed=1234` .

In [75]:
```
## Q1
a = 1
n = 10
c = (3 + a) / (1 - n)
print(c)
```

```
-0.4444444444444444
```

In [81]:
```
## Q2
import math

print(math.factorial(12))
```

```
479001600
```

In [4]:
```
## Q3
import math

mean = 45.6
p = 0.54
n = 1118
standard_error = 1.96 * math.sqrt(p * (1 - p) / n)

# Calculate the lower and upper bounds of the confidence interval
lower_bound = mean - standard_error
upper_bound = mean + standard_error

print("Confidence Interval: [{:.2f}, {:.2f}]".format(lower_bound, upper_bou
```

```
Confidence Interval: [45.57, 45.63]
```

In [6]:
```
## Q4
import math

a = 3
b = 4

c = math.sqrt((a ** 2) + (b ** 2))
print(c)
```

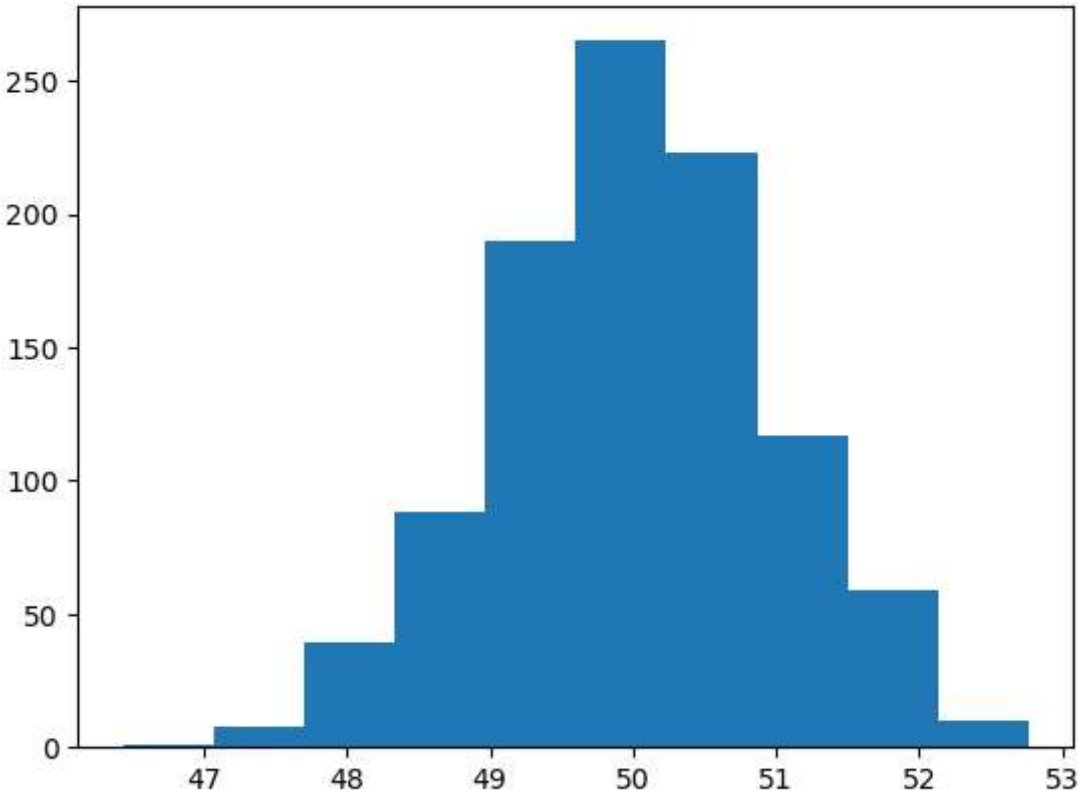
```
5.0
```

In [23]:

```
## Q5
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(1234)
mean = 50
sd = 1
numbers = 1000

data = np.random.normal(mean, sd, numbers)
plt.hist(data)
plt.show()
```

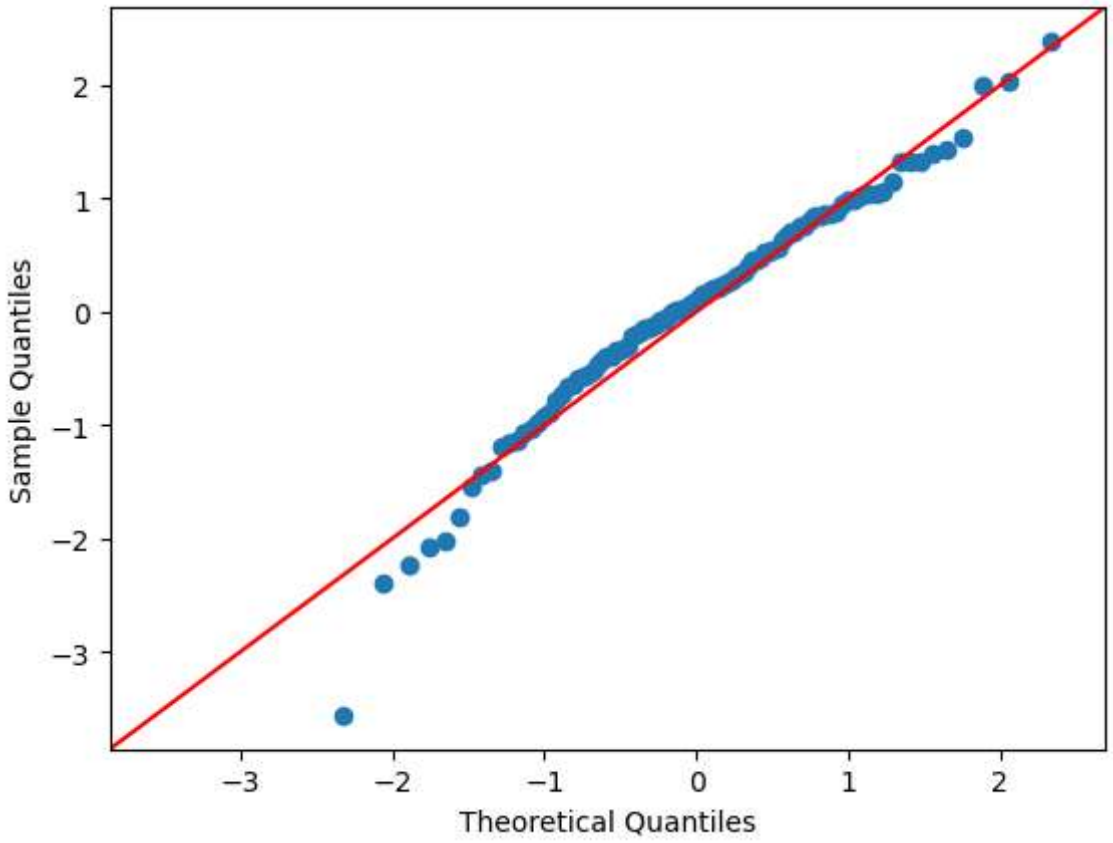


In [25]:

```
## Q6
import matplotlib.pyplot as plt
import numpy as np
import statsmodels.api as sm

np.random.seed(1234)
mean = 0
sd = 1
numbers = 100

data = np.random.normal(mean, sd, numbers)
sm.qqplot(data, line ='45')
plt.show()
```

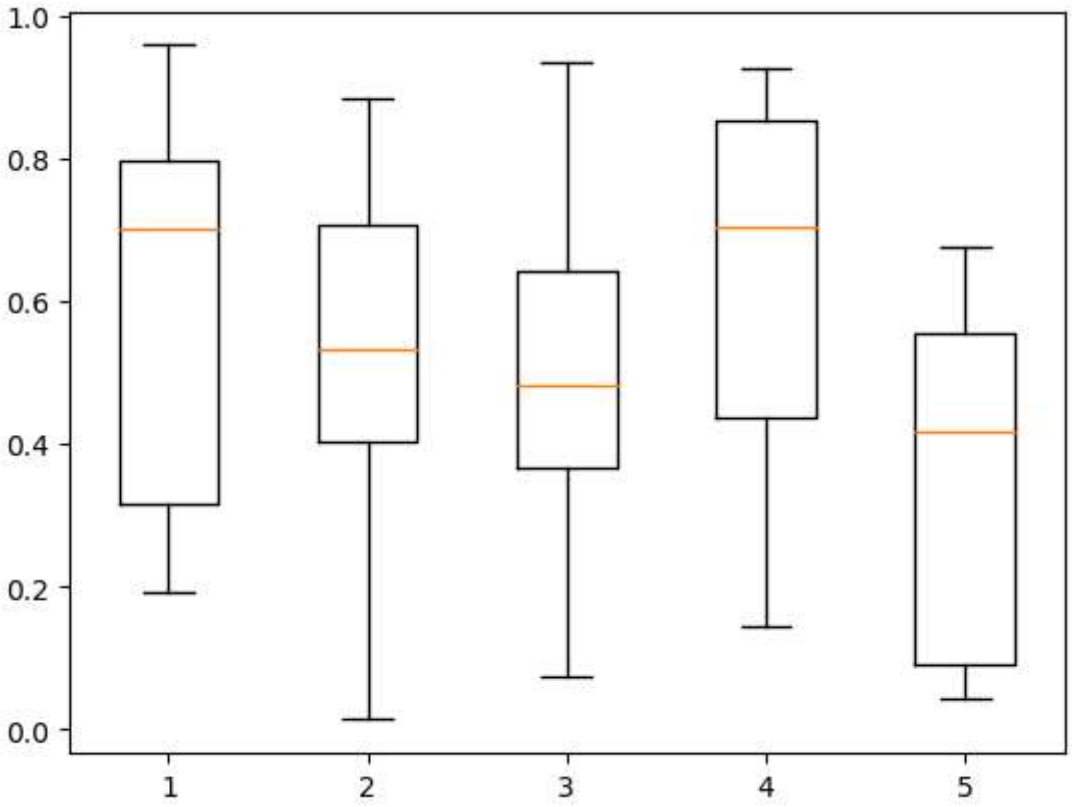


In [58]:
```
## Q7
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(1234)
data = []

for x in range(5):
    data.append(np.random.uniform(0, 1, 10))

plt.boxplot(data)
plt.show()
```



In [62]:
```
## Q8
A = [11,15,17,18,19]

np.std(A)
```

Out[62]: 2.8284271247461903

In [81]:

```
## Q9
from IPython.display import display
from sympy import symbols, Eq, solve
x, y, z, c1 = symbols('x y z c1')

eq1 = Eq(2*(x**2) + y + z - 1, 0)
eq2 = Eq(x + 2*y + z - c1, 0)
eq3 = Eq(-2*x + y + z, 0)

sol = solve((eq1, eq2, eq3),(x, y, z))
display(eq1, eq2, eq3, sol)
```

$$2x^2 + y + z - 1 = 0$$

$$-c_1 + x + 2y + z = 0$$

$$-2x + y + z = 0$$

$$\left[\left(-\frac{1}{2} + \frac{\sqrt{3}}{2},\ c_1 - \frac{3\sqrt{3}}{2} + \frac{3}{2},\ -c_1 - \frac{5}{2} + \frac{5\sqrt{3}}{2}\right),\ \left(-\frac{\sqrt{3}}{2} - \frac{1}{2},\ c_1 + \frac{3}{2} + \frac{3\sqrt{}\right.$$

In [99]:

```
## Q10
import matplotlib.pyplot as plt
import numpy as np

# Method 1
# np.random.seed(1234)
# A = np.linspace(0, 1, num=100)
# sampled_data = np.random.choice(A, size=50)

# plt.hist(sampled_data)
# plt.show()

# Method 2
random.seed(1234)
A = np.linspace(0, 1, num=100)
sampled_data = random.sample(list(A), 50)

plt.hist(sampled_data)
plt.show()
```