Python

Python basics

Kunal Khurana

2023-12-23

Table of contents

Documentation

python methods

sequence types

Lists

Tuples

Functions

1) use the help() funciton to learn about functions
2) invoke a function inside of a function

```
#1
help(round)
```

```
Help on built-in function round in module builtins:

round(number, ndigits=None)
    Round a number to a given precision in decimal digits.

    The return value is an integer if ndigits is omitted or None.  Otherwise
    the return value has the same type as the number.  ndigits may be negative.
```

```
help(round(-2.01))
```

```
Help on int object:

class int(object)
 |  int([x]) -> integer
 |  int(x, base=10) -> integer
 |
 |  Convert a number or string to an integer, or return 0 if no arguments
 |  are given.  If x is a number, return x.__int__().  For floating point
 |  numbers, this truncates towards zero.
 |
 |  If x is not a number or if base is given, then x must be a string,
 |  bytes, or bytearray instance representing an integer literal in the
 |  given base.  The literal can be preceded by '+' or '-' and be surrounded
 |  by whitespace.  The base defaults to 10.  Valid bases are 0 and 2-36.
 |  Base 0 means to interpret the base from the string as an integer literal.
 |  >>> int('0b100', base=0)
```

```
|  4
|
|  Built-in subclasses:
|      bool
|
|  Methods defined here:
|
|  __abs__(self, /)
|      abs(self)
|
|  __add__(self, value, /)
|      Return self+value.
|
|  __and__(self, value, /)
|      Return self&value.
|
|  __bool__(self, /)
|      True if self else False
|
|  __ceil__(...)
|      Ceiling of an Integral returns itself.
|
|  __divmod__(self, value, /)
|      Return divmod(self, value).
|
|  __eq__(self, value, /)
|      Return self==value.
|
|  __float__(self, /)
|      float(self)
|
|  __floor__(...)
|      Flooring an Integral returns itself.
|
|  __floordiv__(self, value, /)
|      Return self//value.
|
|  __format__(self, format_spec, /)
|      Default object formatter.
|
|  __ge__(self, value, /)
|      Return self>=value.
|
```

```
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getnewargs__(self, /)
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __hash__(self, /)
|      Return hash(self).
|
|  __index__(self, /)
|      Return self converted to an integer, if self is suitable for use as an index into a l
|
|  __int__(self, /)
|      int(self)
|
|  __invert__(self, /)
|      ~self
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __lshift__(self, value, /)
|      Return self<<value.
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __mod__(self, value, /)
|      Return self%value.
|
|  __mul__(self, value, /)
|      Return self*value.
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __neg__(self, /)
|      -self
|
|  __or__(self, value, /)
|      Return self|value.
```

```
|
|  __pos__(self, /)
|      +self
|
|  __pow__(self, value, mod=None, /)
|      Return pow(self, value, mod).
|
|  __radd__(self, value, /)
|      Return value+self.
|
|  __rand__(self, value, /)
|      Return value&self.
|
|  __rdivmod__(self, value, /)
|      Return divmod(value, self).
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __rfloordiv__(self, value, /)
|      Return value//self.
|
|  __rlshift__(self, value, /)
|      Return value<<self.
|
|  __rmod__(self, value, /)
|      Return value%self.
|
|  __rmul__(self, value, /)
|      Return value*self.
|
|  __ror__(self, value, /)
|      Return value|self.
|
|  __round__(...)
|      Rounding an Integral returns itself.
|
|      Rounding with an ndigits argument also returns an integer.
|
|  __rpow__(self, value, mod=None, /)
|      Return pow(value, self, mod).
|
|  __rrshift__(self, value, /)
```

```
|      Return value>>self.
|
|  __rshift__(self, value, /)
|      Return self>>value.
|
|  __rsub__(self, value, /)
|      Return value-self.
|
|  __rtruediv__(self, value, /)
|      Return value/self.
|
|  __rxor__(self, value, /)
|      Return value^self.
|
|  __sizeof__(self, /)
|      Returns size in memory, in bytes.
|
|  __sub__(self, value, /)
|      Return self-value.
|
|  __truediv__(self, value, /)
|      Return self/value.
|
|  __trunc__(...)
|      Truncating an Integral returns itself.
|
|  __xor__(self, value, /)
|      Return self^value.
|
|  as_integer_ratio(self, /)
|      Return integer ratio.
|
|      Return a pair of integers, whose ratio is exactly equal to the original int
|      and with a positive denominator.
|
|      >>> (10).as_integer_ratio()
|      (10, 1)
|      >>> (-10).as_integer_ratio()
|      (-10, 1)
|      >>> (0).as_integer_ratio()
|      (0, 1)
|
|  bit_count(self, /)
```

```
|      Number of ones in the binary representation of the absolute value of self.
|
|      Also known as the population count.
|
|      >>> bin(13)
|      '0b1101'
|      >>> (13).bit_count()
|      3
|
|  bit_length(self, /)
|      Number of bits necessary to represent self in binary.
|
|      >>> bin(37)
|      '0b100101'
|      >>> (37).bit_length()
|      6
|
|  conjugate(...)
|      Returns self, the complex conjugate of any int.
|
|  to_bytes(self, /, length=1, byteorder='big', *, signed=False)
|      Return an array of bytes representing an integer.
|
|      length
|        Length of bytes object to use.  An OverflowError is raised if the
|        integer is not representable with the given number of bytes.  Default
|        is length 1.
|      byteorder
|        The byte order used to represent the integer.  If byteorder is 'big',
|        the most significant byte is at the beginning of the byte array.  If
|        byteorder is 'little', the most significant byte is at the end of the
|        byte array.  To request the native byte order of the host system, use
|        `sys.byteorder' as the byte order value.  Default is to use 'big'.
|      signed
|        Determines whether two's complement is used to represent the integer.
|        If signed is False and a negative integer is given, an OverflowError
|        is raised.
|
|  ----------------------------------------------------------------------
|  Class methods defined here:
|
|  from_bytes(bytes, byteorder='big', *, signed=False) from builtins.type
|      Return the integer represented by the given array of bytes.
```

```
|
|      bytes
|        Holds the array of bytes to convert.  The argument must either
|        support the buffer protocol or be an iterable object producing bytes.
|        Bytes and bytearray are examples of built-in objects that support the
|        buffer protocol.
|      byteorder
|        The byte order used to represent the integer.  If byteorder is 'big',
|        the most significant byte is at the beginning of the byte array.  If
|        byteorder is 'little', the most significant byte is at the end of the
|        byte array.  To request the native byte order of the host system, use
|        `sys.byteorder' as the byte order value.  Default is to use 'big'.
|      signed
|        Indicates whether two's complement is used to represent the integer.
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data descriptors defined here:
|
|  denominator
|      the denominator of a rational number in lowest terms
|
|  imag
|      the imaginary part of a complex number
|
|  numerator
|      the numerator of a rational number in lowest terms
|
|  real
|      the real part of a complex number
```

Defining functions

```
def least_difference(a, b, c):
    diff1 = abs(a - b)
    diff2 = abs(b - c)
    diff3 = abs(a - c)
    return min(diff1, diff2, diff3)
```

```
help(least_difference)
```

```
Help on function least_difference in module __main__:

least_difference(a, b, c)
```

```
least_difference (19, 2, 10)
```

```
8
```

Dockstring

```
def least_difference(a, b, c):
    """
    Returns the smallest difference between any two
    numbers among a, b and c.

    e.g- least_difference (19, 2, 10)
    >>> 8

    """
    diff1 = abs(a - b)
    diff2 = abs(b - c)
    diff3 = abs(a - c)
    return min(diff1, diff2, diff3)
```

```
help(least_difference)
```

```
Help on function least_difference in module __main__:

least_difference(a, b, c)
    Returns the smallest difference between any two
    numbers among a, b and c.

    e.g- least_difference (19, 2, 10)
    >>> 8
```

Default argument

```
print (1, 4, 5, sep = ' <')
```

```
1 <4 <5
```

Functions applied to Functions

1. supplying functions as arguments to other functions

```
def mult_by_five(x):
    return 5 * x

def call(fn, arg):
    """Call fn on agr"""
    return fn(arg)

def squared_call(fn, arg):
    """Call fn as a result of calling fn on arg"""
    return fn(fn(arg))

print(
    call(mult_by_five, 1),
    squared_call(mult_by_five, 1),
    sep = '\n',
)
```

```
5
25
```

2. Higher order functions- functions that operate over other functions

```
def mod_5(x):
    """Retuns the remainder of x after dividing by 5"""
    return x % 5

print(
    'Which number is biggest?',
    max(103, 332, 44),
    'Which numbner is the biggest modulo 5?',
    max(103, 332, 44, key = mod_5),
    sep = '\n',
)
```

```
Which number is biggest?
332
Which numbner is the biggest modulo 5?
44
```

```
x = -10
y = 5
#Which of the two variables above has the smallest absolute value?
smallest_abs = min(abs(x, y))
```

```
TypeError: abs() takes exactly one argument (2 given)
```

```
x = -10
y = 5

# Calculate the absolute values
abs_x = abs(x)
abs_y = abs(y)

# Find the minimum absolute value
smallest_abs = min(abs_x, abs_y)

# Print the result
print(f"The smallest absolute value is: {smallest_abs}")
```

```
The smallest absolute value is: 5
```

```
def f(x):
    y = abs(x)
    return y
print(f(5))
```

```
5
```

Booleans

```
x= True
print(x)
print(type(x))
```

```
True
<class 'bool'>
```

```
def is_odd(n):
    return (n % 2) == 1
print(f"Is 45 odd?", is_odd(45))
```

```
Is 45 odd? True
```

```
True or True and False
```

```
True
```

```
True and False  # above expression gets operated from Right to Left
```

```
False
```

```
True or False  # why true or false is true? à voir
```

```
True
```

write simpler code

prepared_for_weather = have_umbrella or rain_level < 5 and have_hood or not rain_level > 0 and is_workday

```
# better way to writing code to make it readable
#prepared_for_weather = (
    have_umbrella
    or ((rain_level < 5) and have_hood)
    or (not (rain_level > 0 and is_workday))
    )
```

Conditionals

```
def inspect(x):
    if x == 0:
        print(x, "is zero")
    elif x > 0:
        print(x, "is positive")
    elif x < 0:
        print(x, "is negative")
    else:
        print(x, "is unlike anything I've ever seen...")

inspect(0)
inspect(-15)
```

```
0 is zero
-15 is negative
```

```
def f(x):
    if x > 0:
        print("Only printed when x is positive; x =", x)
        print("Also only printed when x is positive; x =", x)
    print("Always printed, regardless of x's value; x =", x)

f(1)
f(0)
```

```
Only printed when x is positive; x = 1
Also only printed when x is positive; x = 1
```

```
Always printed, regardless of x's value; x = 1
Always printed, regardless of x's value; x = 0
```

```
print(bool(1)) # all numbers are treated as true, except 0
print(bool(0))
print(bool("asf")) # all strings are treated as true, except the empty string ""
print(bool(""))
# Generally empty sequences (strings, lists, and other types we've yet to see like lists a
# are "falsey" and the rest are "truthy"
```

```
True
False
True
False
```

```
def sign(x):
    if x < 0:
        return (-1)
    elif x > 0:
        return (1)
    else:
        return (0)
sign(45)
```

```
1
```

```
def to_smash(total_candies):
    """Return the number of leftover candies that must be smashed after distributing
    the given number of candies evenly between 3 friends.

    >>> to_smash(91)
    1
    """
    print(f"Splitting", total_candies, "candies")
    return total_candies % 3

to_smash(91)
```

```
Splitting 91 candies
```

```
1
```

```
to_smash(1)
```

```
Splitting 1 candies

1
```

```
def is_negative(number):
    if number < 0:
        return True
    else:
        return False

def concise_is_negative(number):
    return number < 0
    pass
```

```
is_negative(-5)
```

```
True
```

```
concise_is_negative(-4)
```

```
True
```

```
def onionless(ketchup, mustard, onion):
    """Return whether the customer doesn't want onions.
    """
    return not onion
```

```
def wants_plain_hotdog(ketchup, mustard, onion):
    """Return whether the customer wants a plain hot dog with no toppings.
    """
    return not ketchup and not mustard and not onion
```

```
def exactly_one_topping(ketchup, mustard, onion):
    """Return whether the customer wants exactly one of the three available toppings
    on their hot dog.
    """
    return (ketchup + mustard + onion) == 1
```

Lists

```
# returning None if the list contains less than 2 elements

def select_second (L):
    if len(L) < 2:
        return None
    return L [1]
```

```
# test
print(select_second([1, 2]))
print(select_second([0]))
```

```
2
None
```

```
# print the captain of the loosing team. Teams are in descending order
# in the list, coach is first, followed by the captain

def losing_team_captain(teams):
    return teams[-1][1]


teams = [
    ['Coach1', 'Captain1', 'player1', 'player2'],
    ['Coach2', 'Captain2', 'player21', 'player22'],
    ['Coach3', 'Captain3', 'player31', 'player32']
]

print(losing_team_captain(teams))
```

```
Captain3
```

```
def swap_names (racers):
    '''this function swaps the first and last places of the racers'''
    racers[0], racers [-1] = racers[-1], racers[0]
    return racers


racers =['Mario', 'Suzaine', 'Randy']

print(swap_names(racers))
```

```
['Randy', 'Suzaine', 'Mario']
```

```
help(swap_names)
```

```
Help on function swap_names in module __main__:

swap_names(racers)
    this function swaps the first and last places of the racers
```

```
party_attendees = ['Adela', 'Fleda', 'Owen', 'May', 'Mona', 'Gilbert', 'Ford']


def fashionably_late(attendees):
    '''
    print the names of people who arrived after half of the party guests, but not the last
    '''
    # mid point
    mid = (len(attendees) // 2) + 1
    # remaining list
    return attendees[mid:-1]

print(fashionably_late(party_attendees))
```

```
['Mona', 'Gilbert']
```

```
help(fashionably_late)
```

```
Help on function fashionably_late in module __main__:

fashionably_late(attendees)
    print the names of people who arrived after half of the party guests, but not the last
```

Loops and List Comprehensions

```
# way to repeatedly execute the code
planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
for planet in planets:
    print(planet, end=' ')
```

```
Mercury Venus Earth Mars Jupiter Saturn Uranus Neptune
```

For loops

```
multiplicands = (2, 2, 2, 3, 3, 5)
product = 1
for mult in multiplicands:
    product = product * mult
print(product)
```

```
360
```

```
# loop through each character in a string
s = 'steganograpHy is the practicE of conceaLing a file, message, image, or video within a
msg = ''
# print all the uppercase letters in s, one at a time
for char in s:
    if char.isupper():
        print(char, end='')
```

```
HELLO
```

```python
# range function
for i in range(5):
    print(f"Doing important work = ", i)
```

```
Doing important work =  0
Doing important work =  1
Doing important work =  2
Doing important work =  3
Doing important work =  4
```

While loops

```python
# iterates until condition is met
"""write a while loop of numbers from 0 until the sum reaches 50 by continuous additions o
i = 1
while i < 50:
    print(i , end = " ")
```

```
1 2 4 8 16 32
```

List Comprehensions

syntax- [expression for item in list if conditional]

nested list comprehension syntax- [[expression for item in sublist] for sublist in list]

```
squares = [n**2 for n in range(10)]
print(squares)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
## if conditional
'''write list comprehesnion for even numbers upto 10'''
squares = [n**2 for n in range(10) if n%2 == 0 ]
print(squares)
```

```
[0, 4, 16, 36, 64]
```

```
def lucky_number(squares):
"""contains the number 7, let's check that in square module"""
    for num in squares:
        if num% 7 == 0:
            return True
    return False
```

```
lucky_number = [7 for x in squares if x == 7]
print(lucky_number)
```

```
[]
```

```
squares
```

```
[0, 4, 16, 36, 64]
```

```
def elementwise_greater_than(L, thresh):
    """Return a list with the same length as L, where the value at index i is
    True if L[i] is greater than thresh, and False otherwise.

    >>> elementwise_greater_than([1, 2, 3, 4], 2)
    [False, False, True, True]
    """
    return [element > thresh for element in L]

print(elementwise_greater_than([1, 2, 3, 4], 2))
```

```
[False, False, True, True]
```

```
help(elementwise_greater_than)
```

```
Help on function elementwise_greater_than in module __main__:

elementwise_greater_than(L, thresh)
    Return a list with the same length as L, where the value at index i is
    True if L[i] is greater than thresh, and False otherwise.

    >>> elementwise_greater_than([1, 2, 3, 4], 2)
    [False, False, True, True]
```

```
def menu_is_boring(meals):
    """Given a list of meals served over some period of time, return True if the
    same meal has ever been served two days in a row, and False otherwise.
    """
    for i in range(len(meals) - 1):  # Iterate over indices, stopping one short of the end
        if meals[i] == meals[i + 1]:
            return True  # Same meal served two days in a row
    return False  # No meal served two days in a row

# Example usage
print(menu_is_boring(["pasta", "salad", "soup", "salad", "steak"]))
print(menu_is_boring(["pasta", "salad", "soup", "sandwich", "steak"]))
```

```
False
False
```

Strings

```
# learn about string methods
print(dir(str))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__f
```

```
define = 'pluto is a planet'
claim = define.split()
print(claim)
```

```
['pluto', 'is', 'a', 'planet']
```

```
# adding unicode characters
' '.join([word.upper() for word in claim])
```

```
'PLUTO IS A PLANET'
```

```
# string formatting
s = """Pluto is a dwarf plant. It's the {0} planet.
No, it's the {1}.
{0} hehe
{1} haha""".format('8th', '9th')
print(s)
```

```
Pluto is a dwarf plant. It's the 8th planet.
No, it's the 9th.
8th hehe
9th haha
```

Dictionaries

```
# used for mapping keys to values
numbers = {'one' : 1, 'two' : 2, 'three' : 3}
# calling
numbers['one']
```

```
1
```

```
# adding
numbers['eleven'] = 11  # use square brackets
numbers
```

```
{'one': 1, 'two': 2, 'three': 3, 'eleven': 11}
```

```
# change
numbers['one'] = 'Pluto'
numbers
```

```
{'one': 'Pluto', 'two': 2, 'three': 3, 'eleven': 11}
```

```
# calling all keys along with first caps for each
numbers_defined = {a: a[0] for a in numbers}
numbers_defined
```

```
{'one': 'o', 'two': 't', 'three': 't', 'eleven': 'e'}
```

```
print(dir(dict))
```

```
['__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__dir__',
```

```
  # for loops in dictionaries
  for k in numbers:
      print("{} = {}".format(k, numbers[k]))
```

```
one = Pluto
two = 2
three = 3
eleven = 11
```

```
  # get all initials from the dictionary

  for key, value in numbers_defined.items():
      print("{} is equivalent to \"{}\"".format(key.upper(), value.upper()))
```

```
ONE is equivalent to "O"
TWO is equivalent to "T"
THREE is equivalent to "T"
ELEVEN is equivalent to "E"
```

```
  # modern formatting using f-string
  for key, value in numbers_defined.items():
      print(f"{key.upper()} is equivalent to \"{value.upper()}\"")
```

```
ONE is equivalent to "O"
TWO is equivalent to "T"
THREE is equivalent to "T"
ELEVEN is equivalent to "E"
```

```
  def is_valid_zip(zip_code):
      """
      Returns whether the input string is a valid (5 digit) zip code

      Remember, take the value of function to play with it

      example- return len(zip_code) == 5 and zip_code.isdigit()

      """
```

```
help(dir(str))
```

```
Help on list object:

class list(object)
 |  list(iterable=(), /)
 |
 |  Built-in mutable sequence.
 |
 |  If no argument is given, the constructor creates a new empty list.
 |  The argument must be an iterable if specified.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __delitem__(self, key, /)
 |      Delete self[key].
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(...)
 |      x.__getitem__(y) <==> x[y]
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iadd__(self, value, /)
 |      Implement self+=value.
 |
 |  __imul__(self, value, /)
```

```
|      Implement self*=value.
|
|  __init__(self, /, *args, **kwargs)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __mul__(self, value, /)
|      Return self*value.
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __reversed__(self, /)
|      Return a reverse iterator over the list.
|
|  __rmul__(self, value, /)
|      Return value*self.
|
|  __setitem__(self, key, value, /)
|      Set self[key] to value.
|
|  __sizeof__(self, /)
|      Return the size of the list in memory, in bytes.
|
|  append(self, object, /)
|      Append object to the end of the list.
|
|  clear(self, /)
|      Remove all items from list.
```

```
|
|  copy(self, /)
|      Return a shallow copy of the list.
|
|  count(self, value, /)
|      Return number of occurrences of value.
|
|  extend(self, iterable, /)
|      Extend list by appending elements from the iterable.
|
|  index(self, value, start=0, stop=9223372036854775807, /)
|      Return first index of value.
|
|      Raises ValueError if the value is not present.
|
|  insert(self, index, object, /)
|      Insert object before index.
|
|  pop(self, index=-1, /)
|      Remove and return item at index (default last).
|
|      Raises IndexError if list is empty or index is out of range.
|
|  remove(self, value, /)
|      Remove first occurrence of value.
|
|      Raises ValueError if the value is not present.
|
|  reverse(self, /)
|      Reverse *IN PLACE*.
|
|  sort(self, /, *, key=None, reverse=False)
|      Sort the list in ascending order and return None.
|
|      The sort is in-place (i.e. the list itself is modified) and stable (i.e. the
|      order of two equal elements is maintained).
|
|      If a key function is given, apply it once to each list item and sort them,
|      ascending or descending, according to their function values.
|
|      The reverse flag can be set to sort in descending order.
|
|  ----------------------------------------------------------------------
```

```
 |  Class methods defined here:
 |
 |  __class_getitem__(...) from builtins.type
 |      See PEP 585
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __hash__ = None
```

```
help(is_valid_zip)
```

```
Help on function is_valid_zip in module __main__:

is_valid_zip(zip_code)
    Returns whether the input string is a valid (5 digit) zip code

    Remember, take the value of function to play with it

    example- return len(zip_code) == 5 and zip_code.isdigit()
```

```
def word_search(doc_list, keyword):

    """
    Takes a list of documents (each document is a string) and a keyword.
    Returns list of the index values into the original list for all documents
    containing the keyword.

    this involves three steps-
    1. make sure that we get the desired word, not a part of it
    2. upper and lower case letter are eqaul and should be counted
    3. periods or commas should not affect the call.


    """
```

```
help(word_search)
```

```
Help on function word_search in module __main__:

word_search(doc_list, keyword)
    Takes a list of documents (each document is a string) and a keyword.
    Returns list of the index values into the original list for all documents
    containing the keyword.

    this involves three steps-
    1. make sure that we get the desired word, not a part of it
    2. upper and lower case letter are eqaul and should be counted
    3. periods or commas should not affect the call.
```

```
def word_search(doc_list, keyword):
    #create empty list
    empty= []
    # iterate
    for key, value in enumerate(doc_list):
        #split
        seperate = value.split()
        # remove commas and match
        normalize = [a.rstrip(".,").lower() for a in seperate]
        # if there is a match, update the list
        if keyword.lower() in normalize:
            empty.append(key)
    return empty
```

```
# multi word search in a dictionary

def multi_word_serach (documents, keywords):
    keyword_to_indices = {}
    for keyword in keywords:
        keyword_to_indices[keyword]= word_search(documents, keyword)
    return keyword_to_indices
```

Working with external libraries

```
import math
print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'as
```

```
print(f"pi to four decimal places = {math.pi:.4}")
```

```
pi to four decimal places = 3.142
```

```
math.log(34,5)
```

```
2.1910509857959815
```

```
help(math.log)
```

```
Help on built-in function log in module math:

log(...)
    log(x, [base=math.e])
    Return the logarithm of x to the given base.

    If the base not specified, returns the natural logarithm (base e) of x.
```

```
help(math.sin)
```

```
Help on built-in function sin in module math:

sin(x, /)
    Return the sine of x (measured in radians).
```

Submodules

```
import numpy
print ("numpy.random is a", type(numpy.random))
print("it contains names such as....",
      dir(numpy.random)[-15:])
```

```
numpy.random is a <class 'module'>
it contains names such as.... ['set_bit_generator', 'set_state', 'shuffle', 'standard_cauchy
```

```
# roll 10 dice
rolls = numpy.random.randint(low=1, high =6, size = 10)   #Two dots required; for numpy an
rolls
```

```
array([1, 1, 2, 2, 2, 5, 3, 3, 1, 1])
```

```
print(rolls)
```

```
[3 2 1 1 3 4 2 1 3 2]
```

three tools for understaing strange objects

type(),

dir(),

help()

```
type(rolls)
```

```
numpy.ndarray
```

```
print(dir(rolls))
```

```
['T', '__abs__', '__add__', '__and__', '__array__', '__array_finalize__', '__array_function_
```

```
help(rolls.__abs__)
```

```
Help on method-wrapper:

__abs__()
    abs(self)
```

```
help(rolls.prod)
```

```
Help on built-in function prod:

prod(...) method of numpy.ndarray instance
    a.prod(axis=None, dtype=None, out=None, keepdims=False, initial=1, where=True)

    Return the product of the array elements over the given axis

    Refer to `numpy.prod` for full documentation.

    See Also
    --------
    numpy.prod : equivalent function
```

```
rolls + 10
```

```
array([11, 11, 12, 12, 12, 15, 13, 13, 11, 11])
```

```
rolls >= 3
```

```
array([False, False, False, False, False,  True,  True,  True, False,
       False])
```

```
# create a 2-d array

a = [[1,23,4],[3,2,4]]
x = (f"this is how a 2-d array looks like \n{numpy.array(a)}")
```

```
print(x)
```

```
this is how a 2-d array looks like
[[ 1 23  4]
 [ 3  2  4]]
```

```
def blackjack_hand_greater_than(hand_1, hand_2):
    """
    Return True if hand_1 beats hand_2, and False otherwise.

    In order for hand_1 to beat hand_2 the following must be true:
    - The total of hand_1 must not exceed 21
    - The total of hand_1 must exceed the total of hand_2 OR hand_2's total must exceed 21

    Hands are represented as a list of cards. Each card is represented by a string.

    When adding up a hand's total, cards with numbers count for that many points. Face
    cards ('J', 'Q', and 'K') are worth 10 points. 'A' can count for 1 or 11.

    When determining a hand's total, you should try to count aces in the way that
    maximizes the hand's total without going over 21. e.g. the total of ['A', 'A', '9'] is
    the total of ['A', 'A', '9', '3'] is 14.

    Examples:
    >>> blackjack_hand_greater_than(['K'], ['3', '4'])
    True
    >>> blackjack_hand_greater_than(['K'], ['10'])
    False
    >>> blackjack_hand_greater_than(['K', 'K', '2'], ['3'])
    False
    """
```

```
help(blackjack_hand_greater_than)
```

```
Help on function blackjack_hand_greater_than in module __main__:

blackjack_hand_greater_than(hand_1, hand_2)
    Return True if hand_1 beats hand_2, and False otherwise.

    In order for hand_1 to beat hand_2 the following must be true:
```

```
- The total of hand_1 must not exceed 21
- The total of hand_1 must exceed the total of hand_2 OR hand_2's total must exceed 21

Hands are represented as a list of cards. Each card is represented by a string.

When adding up a hand's total, cards with numbers count for that many points. Face
cards ('J', 'Q', and 'K') are worth 10 points. 'A' can count for 1 or 11.

When determining a hand's total, you should try to count aces in the way that
maximizes the hand's total without going over 21. e.g. the total of ['A', 'A', '9'] is 2
the total of ['A', 'A', '9', '3'] is 14.

Examples:
>>> blackjack_hand_greater_than(['K'], ['3', '4'])
True
>>> blackjack_hand_greater_than(['K'], ['10'])
False
>>> blackjack_hand_greater_than(['K', 'K', '2'], ['3'])
False
```

```python
#defining card values, here we've kept A=11
def card_value(card):
    if card in ['J', 'Q', 'K']:
        return 10
    elif card == 'A':
        return 11
    else:
        return int(card)

# totalling one hand and adjusting aces

def hand_total(hand):
    """Calculate the total value of a hand."""
    total = sum(card_value(card) for card in hand)

    #adjust for aces
    aces_count = hand.count('A')
    while total > 21 and aces_count > 0:
        # subtract 10 from total to change an ace from 11 to 1
        total -= 10
        # decrease the count of aces being treated as 11
        aces_count -= 1
```

```
    return total

# defining the comparison operator
def blackjack_hand_greater_than(hand_1, hand_2):
    """return True if hand_1 beats the hand_2 and False otherwise"""
    total_1= hand_total(hand_1)
    total_2= hand_total(hand_2)

    return total_1 <= 21 and (total_1 > total_2 or total_2 > 21)

# Example usage
print(blackjack_hand_greater_than(['K'], ['3', '4']))  # True
print(blackjack_hand_greater_than(['K'], ['10']))  # False
print(blackjack_hand_greater_than(['K', 'K', '2'], ['3']))  # False
```

```
True
False
False
```