In [1]:

```
#20 Operands With Different Operators
operation= 7+22/(50%8)*10+25-(70+83*9)+24//40+92+71*48-(100//114)**18+110/2+50
operation
```

Out[1]:

```
2930.0
```

In [2]:

```
#  capitalize()  Return a capitalized version of the string.
sentence= "lisa did the best she could to draw a map on the small piece of paper."
sentence.capitalize()
```

Out[2]:

```
'Lisa did the best she could to draw a map on the small piece of paper.'
```

In [3]:

```
# casefold()  Return a version of the string suitable for caseless comparisons.
sentence.casefold()
```

Out[3]:

```
'lisa did the best she could to draw a map on the small piece of paper.'
```

In [4]:

```
# center()  Return a centered string of length width.
sentence.center(100,"*")
```

Out[4]:

```
'***************lisa did the best she could to draw a map on the small piece
of paper.***************'
```

In [5]:

```
# count()  Return the number of non-overlapping occurrences of substring sub in
sentence.count("a")
```

Out[5]:

```
6
```

In [6]:

```
# encode()  Encode the string using the codec registered for encoding.
sentence.encode(encoding='utf-16')
```

Out[6]:

```
b'\xff\xfel\x00i\x00s\x00a\x00 \x00d\x00i\x00d\x00 \x00t\x00h\x00e\x00 \x00b
\x00e\x00s\x00t\x00 \x00s\x00h\x00e\x00 \x00c\x00o\x00u\x00l\x00d\x00 \x00t\x
00o\x00 \x00d\x00r\x00a\x00w\x00 \x00a\x00 \x00m\x00a\x00p\x00 \x00o\x00n\x00
\x00t\x00h\x00e\x00 \x00s\x00m\x00a\x00l\x00l\x00 \x00p\x00i\x00e\x00c\x00e\x
00 \x00o\x00f\x00 \x00p\x00a\x00p\x00e\x00r\x00.\x00'
```

In [7]:

```
sentence.encode(encoding='ascii')
```

Out[7]:

```
b'lisa did the best she could to draw a map on the small piece of paper.'
```

In [8]:

```
# endswith()  Return True if S ends with the specified suffix, False otherwise.
sentence.endswith("r")
```

Out[8]:

```
False
```

In [9]:

```
# endswith()  Return True if S ends with the specified suffix, False otherwise.
sentence.endswith(".")
```

Out[9]:

```
True
```

In [10]:

```
# expandtabs()   Return a copy where all tab characters are expanded using spaces.
sentence1="Lisa\t did the best she could to draw a \tmap on the small piece of paper."
sentence1.expandtabs(tabsize=16)
```

Out[10]:

```
'Lisa            did the best she could to draw a               map on the s
mall piece of paper.'
```

In [11]:

```
sentence1.expandtabs(tabsize=-32)
```

Out[11]:

```
'Lisa did the best she could to draw a map on the small piece of paper.'
```

In [12]:

```
# find()  Return the lowest index in S where substring sub is found
sentence= "lisa did the best she could to draw a map on the small piece of paper {}"
sentence.find("a")
```

Out[12]:

```
3
```

In [13]:

```
# format()  Return a formatted version of S, using substitutions from args and kwargs.
sentence.format("using colors")
```

Out[13]:

```
'lisa did the best she could to draw a map on the small piece of paper using
colors'
```

In [14]:

```
# format_map() Return a formatted version of S, using substitutions from mapping.
 #str.format(**mapping) works for Python Dictionaries.
occuption={"x":["Ali","Faisal","Ahmad"],"y":["Engineer","Doctor","Student"]}
print('{x[0]} has {y[0]} profession' .format_map(occuption))
print("{x[1]} has {y[1]} profession".format_map(occuption))
print("{x[2]} has {y[2]} profession".format_map(occuption))
```

```
Ali has Engineer profession
Faisal has Doctor profession
Ahmad has Student profession
```

In [15]:

```
# index()  Return the lowest index in S where substring sub is found,
sentence= "lisa did the best she could to draw a map on the small piece of paper ."
sentence.index("best")
```

Out[15]:

```
13
```

In [16]:

```
# isanum() Return True if the string is an alpha-numeric string, False otherwise.
sentence.isalnum()
```

Out[16]:

```
False
```

In [17]:

```
# isanum() Return True if the string is an alpha-numeric string, False otherwise.
alphanumeric= "Lisa127hascolors56"
alphanumeric.isalnum()
```

Out[17]:

```
True
```

In [18]:

```
# isalpha() Return True if the string is an alphabetic string, False otherwise.
sentence= "lisa did the best she could to draw a map on the small piece of paper ."
sentence.isalpha()
```

Out[18]:

```
False
```

In [19]:

```
# isalpha() Return True if the string is an alphabetic string, False otherwise.
alpha="lisadidthebest"
alpha.isalpha()
```

Out[19]:

```
True
```

In [20]:

```
# isascii() Return True if all characters in the string are ASCII, False otherwise.
sentence= "lisa did the best she could to draw a map on the small piece of paper ."
sentence.isascii()
```

Out[20]:

```
True
```

In [21]:

```
# isascii() Return True if all characters in the string are ASCII, False otherwise.
sentence3= "lisa  did the best she cöuld tö draw a map ön the small piece öf paper ."
sentence3.isascii()
```

Out[21]:

```
False
```

In [22]:

```
# isdecimal() Return True if the string is a decimal string, False otherwise.
sentence.isdecimal()
```

Out[22]:

```
False
```

In [23]:

```
# isdecimal() Return True if the string is a decimal string, False otherwise.
dec="104585527"
dec.isdecimal()
```

Out[23]:

```
True
```

In [24]:

```
# isdigit() Return True if the string is a digit string, False otherwise.
sentence.isdigit()
```

Out[24]:

```
False
```

In [25]:

```
# isdigit() Return True if the string is a digit string, False otherwise.
dec.isdigit()
```

Out[25]:

```
True
```

In [26]:

```
# isidentifier() Return True if the string is a valid Python identifier, False otherwise.
sentence.isidentifier()
```

Out[26]:

```
False
```

In [27]:

```
identify="abcdef"
identify.isidentifier()
```

Out[27]:

```
True
```

In [28]:

```
# islower() Return True if the string is a lowercase string, False otherwise.
sentence.islower()
```

Out[28]:

```
True
```

In [29]:

```
# isnumeric() Return True if the string is a numeric string, False otherwise.
sentence.isnumeric()
```

Out[29]:

```
False
```

In [30]:

```
# isnumeric() Return True if the string is a numeric string, False otherwise.
dec.isnumeric()
```

Out[30]:

```
True
```

In [31]:

```
# isprintable() Return True if the string is printable, False otherwise.
sentence.isprintable()
```

Out[31]:

```
True
```

In [32]:

```
# isprintable() Return True if the string is printable, False otherwise.
sentence1="Lisa\t did the best she could to draw a \tmap on the small piece of paper."
sentence1.isprintable()
```

Out[32]:

```
False
```

In [33]:

```
# isspace() Return True if the string is a whitespace string, False otherwise.
sentence.isspace()
```

Out[33]:

```
False
```

In [34]:

```
# isspace() Return True if the string is a whitespace string, False otherwise.
# This function is used to check if the argument contains all whitespace characters such a
s :
#‘ ‘ – Space
#‘\t’ – Horizontal tab
#‘\n’ – Newline
#‘\v’ – Vertical tab
#‘\f’ – Feed
#‘\r’ – Carriage return
spaceStr="\n "
spaceStr.isspace()
```

Out[34]:

```
True
```

In [35]:

```
#istitle() Return True if the string is a title-cased string, False otherwise.
sentence.istitle()
```

Out[35]:

```
False
```

In [36]:

```
#istitle() Return True if the string is a title-cased string, False otherwise.
titleStr="This Example Is To Check The Title Of String"
titleStr.istitle()
```

Out[36]:

```
True
```

In [37]:

```
# isupper() Return True if the string is an uppercase string, False otherwise.
sentence.isupper()
```

Out[37]:

```
False
```

In [38]:

```
upperStr="HAPPY BIRTHDAY"
upperStr.isupper()
```

Out[38]:

```
True
```

In [39]:

```
# join() Concatenate any number of strings.
identify.join(upperStr)
```

Out[39]:

```
'HabcdefAabcdefPabcdefPabcdefYabcdef abcdefBabcdefIabcdefRabcdefTabcdefHabcde
fDabcdefAabcdefY'
```

In [40]:

```
# ljust() Return a left-justified string of length width.
upperStr.ljust(30,"|")
```

Out[40]:

```
'HAPPY BIRTHDAY||||||||||||||||'
```

In [41]:

```
# lower() Return a copy of the string converted to lowercase.
upperStr.lower()
```

Out[41]:

```
'happy birthday'
```

In [42]:

```
# lstrip() Return a copy of the string with leading whitespace removed.
remStr="\n \n \n Lisa is best \n \n \n"
remStr.lstrip()
```

Out[42]:

```
'Lisa is best \n \n \n'
```

In [43]:

```
#maketrans() Return a translation table usable for str.translate(). It works on dictionary
string="ABC"
string.maketrans(occuption)
```

Out[43]:

```
{120: ['Ali', 'Faisal', 'Ahmad'], 121: ['Engineer', 'Doctor', 'Student']}
```

In [44]:

```
# partition() Partition the string into three parts using the given separator. choose any
 word from string as a separtor
sentence.partition('the')
```

Out[44]:

```
('lisa did ',
 'the',
 ' best she could to draw a map on the small piece of paper .')
```

In [45]:

```
#replace() Return a copy with all occurrences of substring old replaced by new. count mean
s how many times want to replace
sentence.replace("a","o",4)
```

Out[45]:

```
'liso did the best she could to drow o mop on the small piece of paper .'
```

In [46]:

```
# rfind() Return the highest index in S where substring sub is found,
sentence.rfind("a")
```

Out[46]:

```
65
```

In [47]:

```
# rindex() Return the highest index in S where substring sub is found,
sentence.rindex("best")
```

Out[47]:

```
13
```

In [48]:

```
# rjust() Return a right-justified string of length width.
sentence.rjust(80,"$")
```

Out[48]:

```
'$$$$$$$$$lisa did the best she could to draw a map on the small piece of pap
er .'
```

In [49]:

```
# rpartition() Partition the string into three parts using the given separator.
sentence.rpartition("a")
```

Out[49]:

```
('lisa did the best she could to draw a map on the small piece of p',
 'a',
 'per .')
```

In [50]:

```
# rsplit() Return a list of the words in the string, using sep as the delimiter string.
sentence.rsplit(' ', 3)
```

Out[50]:

```
['lisa did the best she could to draw a map on the small piece',
 'of',
 'paper',
 '.']
```

In [51]:

```
# rstrip() Return a copy of the string with trailing whitespace removed.
remStr.rstrip()
```

Out[51]:

```
'\n \n \n Lisa is best'
```

In [52]:

```
# split() Return a list of the words in the string, using sep as the delimiter string.
sentence.split("the")
```

Out[52]:

```
['lisa did ', ' best she could to draw a map on ', ' small piece of paper .']
```

In [53]:

```
# splitlines() Return a list of the lines in the string, breaking at line boundaries.
remStr="\n \n \n Lisa is best \n \n \n"
remStr.splitlines()
```

Out[53]:

```
['', ' ', ' ', ' Lisa is best ', ' ', ' ']
```

In [54]:

```
# startswith() Return True if S starts with the specified prefix, False otherwise.
sentence.startswith("l")
```

Out[54]:

```
True
```

In [55]:

```
# startswith() Return True if S starts with the specified prefix, False otherwise.
sentence.startswith("L")
```

Out[55]:

```
False
```

In [56]:

```
# strip() Return a copy of the string with leading and trailing whitespace remove.
remStr="\n \n \n Lisa is best \n \n \n"
remStr.strip()
```

Out[56]:

```
'Lisa is best'
```

In [57]:

```
# swapcase() Convert uppercase characters to lowercase and lowercase characters to upperca
s
sentence.swapcase()
```

Out[57]:

```
'LISA DID THE BEST SHE COULD TO DRAW A MAP ON THE SMALL PIECE OF PAPER .'
```

In [58]:

```
# title() Return a version of the string where each word is titlecased.
sentence.title()
```

Out[58]:

```
'Lisa Did The Best She Could To Draw A Map On The Small Piece Of Paper .'
```

In [59]:

```
# translate() Replace each character in the string using the given translation table.
str1= "the"
str2="des"
trans1=sentence.maketrans(str1,str2)
sentence.translate(trans1)
```

Out[59]:

```
'lisa did des bssd ses could do draw a map on des small piscs of papsr .'
```

In [60]:

```
# upper() Return a copy of the string converted to uppercase.
sentence.upper()
```

Out[60]:

```
'LISA DID THE BEST SHE COULD TO DRAW A MAP ON THE SMALL PIECE OF PAPER .'
```

In [61]:

```
# zfill() Pad a numeric string with zeros on the left, to fill a field of the given width.
num2="2525"
num2.zfill(6)
```

Out[61]:

```
'002525'
```

In [ ]: