1. Introduction
2. Numeric Datatypes
3. input() and print()
4. control statements--> if,else,elif
5. loops-->while,for

Strings:

1. " " or ' '(characters)--> 'a'
2. immutable --> we cannot make any changes
3. collection of characters enclosed by quotation marks -->iterable objects

In [1]:

```
s1 = "Hello"
s2 = 'Good Afternoon'
print(type(s1),type(s2))
```

```
<class 'str'> <class 'str'>
```

In [3]:

```
s1 = input("enter the string ")
s2 = 'Good Afternoon'
print(type(s1),type(s2))
```

```
enter the string Hello
<class 'str'> <class 'str'>
```

In [15]:

```
#operators : +,*(repitition),[](slice),[start:stop](slice range)
print(s1+s2) #Concatenation
print(s1*3) #Repitition operator
print(s1[4])#slicing
print(s1[-1])#slicing
print(s1[2:]) #star=0,stop=length of the string,step=1
```

```
HelloGood Afternoon
HelloHelloHello
o
o
llo
```

In [17]:

```
print(s1[0:5])
print(s1[:])
```

```
Hello
Hello
```

In [18]:

```
#print alternate values of string
print(s1[::2])
```

Hlo

In [21]:

```
#Accept a string from the user and check whether it is palindrome or not
s3=input("enter the string ")
s3[::-1]
if s3==s3[::-1]:
    print("Palindrome")
else:
    print("Not Palindrome")
```

```
enter the string mam
Palindrome
```

In [22]:

```
n= int(input("Enter a number"))
if str(n) == str(n)[::-1]:
    print("Palindrome")
else:
    print("Not Palindrome")
```

```
Enter a number121
Palindrome
```

In [23]:

```
s4 = "python"
print("on" in s4)
print("xyz" in s4)
print("on" not in s4)
print("xyz" not in s4)
```

```
True
False
False
True
```

Built-in Functions

1. len() --> length of the string
2. max()
3. min()
4. str() --> converting any datatype to string

In [24]:

```
s= "hello"
s1="abc123"
print(len(s))
print(max(s))
print(min(s))
print(len(s1))
print(max(s1))
print(min(s1))
```

```
5
o
e
```

```
6
c
1
```

In [25]:

```python
ord('a')
```

Out[25]:

```
97
```

In [26]:

```python
chr(97)
```

Out[26]:

```
'a'
```

Build-in Methods:

1. capitalize()
2. isalpha() --> returns true when all the characters are alphabets or else returns false
3. isdigit()
4. isalnum()
5. isupper()
6. islower()
7. upper()
8. lower()
9. swapcase()

In [31]:

```python
s="hello"
print(len(s))#sunction_name(object_name)
print(s.capitalize())#object_name.method_name()
print(s)
```

```
5
Hello
hello
```

In [32]:

```python
s1="hello"
s2="abc123"
s3="123"
print(s1.isalpha())
print(s1.isdigit())
print(s1.isalnum())
print(s2.isalpha())
print(s2.isdigit())
print(s2.isalnum())
print(s3.isalpha())
print(s3.isdigit())
print(s3.isalnum())
```

```
True
False
```

```
True
False
False
True
False
True
True
```

In [34]:

```
help("str")
```

```
Help on class str in module builtins:

class str(object)
 |  str(object='') -> str
 |  str(bytes_or_buffer[, encoding[, errors]]) -> str
 |
 |  Create a new string object from the given object. If encoding or
 |  errors is specified, then the object must expose a data buffer
 |  that will be decoded using the given encoding and error handler.
 |  Otherwise, returns the result of object.__str__() (if defined)
 |  or repr(object).
 |  encoding defaults to sys.getdefaultencoding().
 |  errors defaults to 'strict'.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __format__(self, format_spec, /)
 |      Return a formatted version of the string as described by format_spec.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(self, key, /)
 |      Return self[key].
 |
 |  __getnewargs__(...)
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mod__(self, value, /)
 |      Return self%value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __rmod__(self, value, /)
 |      Return value%self.
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  __sizeof__(self, /)
 |      Return the size of the string in memory, in bytes.
 |
 |  __str__(self, /)
 |      Return str(self).
 |
 |  capitalize(self, /)
 |      Return a capitalized version of the string.
 |
 |      More specifically, make the first character have upper case and the rest lower
 |      case.
 |
 |  casefold(self, /)
 |      Return a version of the string suitable for caseless comparisons.
 |
 |  center(self, width, fillchar=' ', /)
 |      Return a centered string of length width.
 |
 |      Padding is done using the specified fill character (default is a space).
 |
 |  count(...)
 |      S.count(sub[, start[, end]]) -> int
 |
 |      Return the number of non-overlapping occurrences of substring sub in
 |      string S[start:end].  Optional arguments start and end are
 |      interpreted as in slice notation.
 |
 |  encode(self, /, encoding='utf-8', errors='strict')
 |      Encode the string using the codec registered for encoding.
 |
 |      encoding
 |        The encoding in which to encode the string.
 |      errors
 |        The error handling scheme to use for encoding errors.
 |        The default is 'strict' meaning that encoding errors raise a
 |        UnicodeEncodeError.  Other possible values are 'ignore', 'replace' and
 |        'xmlcharrefreplace' as well as any other name registered with
 |      codecs.register_error that can handle UnicodeEncodeErrors.
 |
 |  endswith(...)
 |      S.endswith(suffix[, start[, end]]) -> bool
 |
 |      Return True if S ends with the specified suffix, False otherwise.
 |      With optional start, test S beginning at that position.
 |      With optional end, stop comparing S at that position.
 |      suffix can also be a tuple of strings to try.
 |
 |  expandtabs(self, /, tabsize=8)
 |      Return a copy where all tab characters are expanded using spaces.
 |
 |      If tabsize is not given, a tab size of 8 characters is assumed.
 |
 |  find(...)
 |      S.find(sub[, start[, end]]) -> int
 |
 |      Return the lowest index in S where substring sub is found,
 |      such that sub is contained within S[start:end].  Optional
 |      arguments start and end are interpreted as in slice notation.
 |
 |      Return -1 on failure.
 |
 |  format(...)
 |      S.format(*args, **kwargs) -> str
 |
 |      Return a formatted version of S, using substitutions from args and kwargs.
 |      The substitutions are identified by braces ('{' and '}').
 |
 |  format_map(...)
 |      S.format_map(mapping) -> str
 |
 |      Return a formatted version of S, using substitutions from mapping.
 |      The substitutions are identified by braces ('{' and '}').
 |
 |  index(...)
 |      S.index(sub[, start[, end]]) -> int
 |
 |      Return the lowest index in S where substring sub is found,
 |      such that sub is contained within S[start:end].  Optional
 |      arguments start and end are interpreted as in slice notation.
 |
 |      Raises ValueError when the substring is not found.
 |
 |  isalnum(self, /)
 |      Return True if the string is an alpha-numeric string, False otherwise.
 |
 |      A string is alpha-numeric if all characters in the string are alpha-numeric and
 |      there is at least one character in the string.
 |
 |  isalpha(self, /)
 |      Return True if the string is an alphabetic string, False otherwise.
 |
 |      A string is alphabetic if all characters in the string are alphabetic and there
 |      is at least one character in the string.
 |
 |  isascii(self, /)
 |      Return True if all characters in the string are ASCII, False otherwise.
 |
 |      ASCII characters have code points in the range U+0000-U+007F.
 |      Empty string is ASCII too.
 |
 |  isdecimal(self, /)
 |      Return True if the string is a decimal string, False otherwise.
 |
 |      A string is a decimal string if all characters in the string are decimal and
 |      there is at least one character in the string.
 |
 |  isdigit(self, /)
 |      Return True if the string is a digit string, False otherwise.
 |
 |      A string is a digit string if all characters in the string are digits and there
 |      is at least one character in the string.
 |
 |  isidentifier(self, /)
 |      Return True if the string is a valid Python identifier, False otherwise.
 |
 |      Use keyword.iskeyword() to test for reserved identifiers such as "def" and
 |      "class".
 |
 |  islower(self, /)
 |      Return True if the string is a lowercase string, False otherwise.
 |
 |      A string is lowercase if all cased characters in the string are lowercase and
 |      there is at least one cased character in the string.
 |
 |  isnumeric(self, /)
 |      Return True if the string is a numeric string, False otherwise.
 |
 |      A string is numeric if all characters in the string are numeric and there is at
 |      least one character in the string.
 |
 |  isprintable(self, /)
 |      Return True if the string is printable, False otherwise.
 |
 |      A string is printable if all of its characters are considered printable in
 |      repr() or if it is empty.
 |
 |  isspace(self, /)
 |      Return True if the string is a whitespace string, False otherwise.
 |
 |      A string is whitespace if all characters in the string are whitespace and there
 |      is at least one character in the string.
 |
 |  istitle(self, /)
 |      Return True if the string is a title-cased string, False otherwise.
 |
 |      In a title-cased string, upper- and title-case characters may only
 |      follow uncased characters and lowercase characters only cased ones.
 |
 |  isupper(self, /)
 |      Return True if the string is an uppercase string, False otherwise.
 |
 |      A string is uppercase if all cased characters in the string are uppercase and
 |      there is at least one cased character in the string.
 |
 |  join(self, iterable, /)
 |      Concatenate any number of strings.
 |
 |      The string whose method is called is inserted in between each given string.
 |      The result is returned as a new string.
 |
 |      Example: '.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'
 |
 |  ljust(self, width, fillchar=' ', /)
 |      Return a left-justified string of length width.
 |
 |      Padding is done using the specified fill character (default is a space).
 |
 |  lower(self, /)
 |      Return a copy of the string converted to lowercase.
 |
 |  lstrip(self, chars=None, /)
 |      Return a copy of the string with leading whitespace removed.
 |
 |      If chars is given and not None, remove characters in chars instead.
 |
 |  partition(self, sep, /)
 |      Partition the string into three parts using the given separator.
 |
 |      This will search for the separator in the string.  If the separator is found,
 |      returns a 3-tuple containing the part before the separator, the separator
 |      itself, and the part after it.
 |
 |      If the separator is not found, returns a 3-tuple containing the original string
 |      and two empty strings.
 |
 |  replace(self, old, new, count=-1, /)
 |      Return a copy with all occurrences of substring old replaced by new.
 |
 |        count
 |          Maximum number of occurrences to replace.
 |          -1 (the default value) means replace all occurrences.
 |
 |      If the optional argument count is given, only the first count occurrences are
 |      replaced.
 |
 |  rfind(...)
 |      S.rfind(sub[, start[, end]]) -> int
 |
 |      Return the highest index in S where substring sub is found,
 |      such that sub is contained within S[start:end].  Optional
 |      arguments start and end are interpreted as in slice notation.
 |
 |      Return -1 on failure.
 |
 |  rindex(...)
 |      S.rindex(sub[, start[, end]]) -> int
 |
 |      Return the highest index in S where substring sub is found,
 |      such that sub is contained within S[start:end].  Optional
 |      arguments start and end are interpreted as in slice notation.
 |
 |      Raises ValueError when the substring is not found.
 |
 |  rjust(self, width, fillchar=' ', /)
 |      Return a right-justified string of length width.
 |
 |      Padding is done using the specified fill character (default is a space).
 |
 |  rpartition(self, sep, /)
 |      Partition the string into three parts using the given separator.
 |
 |      This will search for the separator in the string, starting at the end. If
 |      the separator is found, returns a 3-tuple containing the part before the
 |      separator, the separator itself, and the part after it.
 |
 |      If the separator is not found, returns a 3-tuple containing two empty strings
 |      and the original string.
 |
 |  rsplit(self, /, sep=None, maxsplit=-1)
 |      Return a list of the words in the string, using sep as the delimiter string.
 |
 |        sep
 |          The delimiter according which to split the string.
 |          None (the default value) means split according to any whitespace,
 |          and discard empty strings from the result.
 |        maxsplit
 |          Maximum number of splits to do.
 |          -1 (the default value) means no limit.
 |
 |      Splits are done starting at the end of the string and working to the front.
 |
 |  rstrip(self, chars=None, /)
 |      Return a copy of the string with trailing whitespace removed.
 |
 |      If chars is given and not None, remove characters in chars instead.
 |
 |  split(self, /, sep=None, maxsplit=-1)
 |      Return a list of the words in the string, using sep as the delimiter string.
 |
 |      sep
 |        The delimiter according which to split the string.
 |        None (the default value) means split according to any whitespace,
 |        and discard empty strings from the result.
 |      maxsplit
 |        Maximum number of splits to do.
 |        -1 (the default value) means no limit.
 |
 |  splitlines(self, /, keepends=False)
 |      Return a list of the lines in the string, breaking at line boundaries.
 |
 |      Line breaks are not included in the resulting list unless keepends is given and
 |      true.
 |
 |  startswith(...)
 |      S.startswith(prefix[, start[, end]]) -> bool
 |
 |      Return True if S starts with the specified prefix, False otherwise.
 |      With optional start, test S beginning at that position.
 |      With optional end, stop comparing S at that position.
 |      prefix can also be a tuple of strings to try.
 |
 |  strip(self, chars=None, /)
 |      Return a copy of the string with leading and trailing whitespace removed.
 |
 |      If chars is given and not None, remove characters in chars instead.
 |
 |  swapcase(self, /)
 |      Convert uppercase characters to lowercase and lowercase characters to uppercase.
 |
 |  title(self, /)
 |      Return a version of the string where each word is titlecased.
 |
 |      More specifically, words start with uppercased characters and all remaining
 |      cased characters have lower case.
 |
 |  translate(self, table, /)
 |      Replace each character in the string using the given translation table.
 |      
 |        table
 |          Translation table, which must be a mapping of Unicode ordinals to
 |          Unicode ordinals, strings, or None.
 |      
 |      The table must implement lookup/indexing via __getitem__, for instance a
 |      dictionary or list.  If this operation raises LookupError, the character is
 |      left untouched.  Characters mapped to None are deleted.
 |  
 |  upper(self, /)
 |      Return a copy of the string converted to uppercase.
 |  
 |  zfill(self, width, /)
 |      Pad a numeric string with zeros on the left, to fill a field of the given width.
 |      
 |      The string is never truncated.
 |  
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |  
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |  
 |  maketrans(x, y=None, z=None, /)
 |      Return a translation table usable for str.translate().
 |      
 |      If there is only one argument, it must be a dictionary mapping Unicode
 |      ordinals (integers) or characters to Unicode ordinals, strings or None.
 |      Character keys will be then converted to ordinals.
 |      If there are two arguments, they must be strings of equal length, and
 |      in the resulting dictionary, each character in x will be mapped to the
 |      character at the same position in y. If there is a third argument, it
 |      must be a string, whose characters will be mapped to None in the result.
```

In [40]:

```
s1="python"
s2="PYTHON"
print(s1.isupper())
print(s2.isupper())
print(s1.islower())
print(s2.islower())
```

```
False
True
True
False
```

In [41]:

```
print(s1.upper())
print(s2.lower())
print(s1.swapcase())
```

```
PYTHON
python
PYTHON
```

In [42]:

```
s="PYthON"
print(s.swapcase())
```

```
pyTHon
```

In [43]:

```
s="abc abc abc"
s.count("abc") #count returns the frequency of the given substring
```

Out[43]:

```
3
```

In [45]:

```
#split() --> return type is list
print(s.split())
s1="a,b,c,d"
print(s1.split(","))
```

```
['abc', 'abc', 'abc']
['a', 'b', 'c', 'd']
```

In [48]:

```
#replace() -->
s="abc" #azc
#s[1]='z'(throws an error)
#s.replace('b','z') o/p:abc (can't change becoz string is immutable)
s=s.replace('b','z')
print(s)
```

```
azc
```

In [52]:

```
# accept a number from the userand remove nth position character from the string
n=int(input("enter position: "))
s=input("enter a string: ")
print(s.replace(s[n-1],"",1))
```

```
enter position: 3
enter a string: abcabc
ababc
```

for loop:

1. for loop whith range() function
2. for loop with iterable object

for variable_name in range(start,stop,step):
  statements

In [55]:

```
#print 1-n numbers on the screen
n=10
for i in range(1,n+1):
    print(i,end=" ")

print()

i=1
while i<=n:
    print(i,end=" ")
    i+=1
```

```
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
```

```
for variable in iterable obj:
    statements
```

In [65]:

```
s="python"
for i in s:
    print(i)
#using while loop print all the characters of the string vertically
i=0
while (i<len(s)):
    print(s[i])
    i+=1
#for loop without using iterable object
for j in range(0,len(s)):
    print(s[j])
```

```
p
y
t
h
o
n
p
y
t
h
o
n
p
y
t
h
o
n
```

1. Accept a string the user and count no.of vowels,consonants and special charaters available in string.
2. count the no.of pairs of a in the given string string --> "abbaaccbbaaa" output --> 2
3. Test case1:

    string-->"((((()()())))))()"

    output-->8 Test case2: string-->"((()))()" Output-->4

In [ ]:

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```