Doc Strings

Print to PDF

Documentation is one of the key aspect related to programming. However, it should be crisp and informative. In Python we can use doc strings for the documentation of our code.

- One of the key aspect of documentation is to provide information about usage of a function.
- In Python we can get the information about the function by using help.
- We can get help for a class like `str` using `help(str)` and help for a function like `str.startswith` using `help(str.startswith)`.
- If you want to provide help for user defined function, you can leverage the feature of Doc Strings. It is nothing but a string which is provided as first statement in a function.
- Here are some of the characteristics related to Doc Strings:
  - By default help returns the function specification.
  - Doc String should be the first line in the function body.
  - The Doc String should not be assigned to any variable.
  - Using `"""` or `'''`, we can have multi-line string.
- It is a good practice to provide crisp and concise Doc String for each of the custom function developed.

```
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__(...)
 |      S.__format__(format_spec) -> str
 |      
 |      Return a formatted version of S 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.
 |  
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |  
 |  __repr__(self, /)
 |      Return repr(self).
 |  
 |  __rmod__(self, value, /)
 |      Return value%self.
 |  
 |  __rmul__(self, value, /)
 |      Return value*self.
 |  
 |  __sizeof__(...)
 |      S.__sizeof__() -> size of S in memory, in bytes
 |  
 |  __str__(self, /)
 |      Return str(self).
 |  
 |  capitalize(...)
 |      S.capitalize() -> str
 |
 |      Return a capitalized version of S, i.e. make the first character
 |      have upper case and the rest lower case.
 |
 |  casefold(...)
 |      S.casefold() -> str
 |
 |      Return a version of S suitable for caseless comparisons.
 |
 |  center(...)
 |      S.center(width[, fillchar]) -> str
 |
 |      Return S centered in a 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(...)
 |      S.encode(encoding='utf-8', errors='strict') -> bytes
 |
 |      Encode S using the codec registered for encoding. Default encoding
 |      is 'utf-8'. errors may be given to set a different error
 |      handling scheme. 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(...)
 |      S.expandtabs(tabsize=8) -> str
 |
 |      Return a copy of S 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(...)
 |      S.isalnum() -> bool
 |
 |      Return True if all characters in S are alphanumeric
 |      and there is at least one character in S, False otherwise.
 |
 |  isalpha(...)
 |      S.isalpha() -> bool
 |
 |      Return True if all characters in S are alphabetic
 |      and there is at least one character in S, False otherwise.
 |
 |  isdecimal(...)
 |      S.isdecimal() -> bool
 |      
 |      Return True if there are only decimal characters in S,
 |      False otherwise.
 |  
 |  isdigit(...)
 |      S.isdigit() -> bool
 |      
 |      Return True if all characters in S are digits
 |      and there is at least one character in S, False otherwise.
 |  
 |  isidentifier(...)
 |      S.isidentifier() -> bool
 |      
 |      Return True if S is a valid identifier according
 |      to the language definition.
 |      
 |      Use keyword.iskeyword() to test for reserved identifiers
 |      such as "def" and "class".
 |  
 |  islower(...)
 |      S.islower() -> bool
 |      
 |      Return True if all cased characters in S are lowercase and there is
 |      at least one cased character in S, False otherwise.
 |  
 |  isnumeric(...)
 |      S.isnumeric() -> bool
 |      
 |      Return True if there are only numeric characters in S,
 |      False otherwise.
 |  
 |  isprintable(...)
 |      S.isprintable() -> bool
 |      
 |      Return True if all characters in S are considered
 |      printable in repr() or S is empty, False otherwise.
 |  
 |  isspace(...)
 |      S.isspace() -> bool
 |      
 |      Return True if all characters in S are whitespace
 |      and there is at least one character in S, False otherwise.
 |  
 |  istitle(...)
 |      S.istitle() -> bool
 |      
 |      Return True if S is a titlecased string and there is at least one
 |      character in S, i.e. upper- and titlecase characters may only
 |      follow uncased characters and lowercase characters only cased ones.
 |      Return False otherwise.
 |  
 |  isupper(...)
 |      S.isupper() -> bool
 |      
 |      Return True if all cased characters in S are uppercase and there is
 |      at least one cased character in S, False otherwise.
 |  
 |  join(...)
 |      S.join(iterable) -> str
 |      
 |      Return a string which is the concatenation of the strings in the
 |      iterable.  The separator between elements is S.
 |  
 |  ljust(...)
 |      S.ljust(width[, fillchar]) -> str
 |      
 |      Return S left-justified in a Unicode string of length width. Padding is
 |      done using the specified fill character (default is a space).
 |  
 |  lower(...)
 |      S.lower() -> str
 |      
 |      Return a copy of the string S converted to lowercase.
 |  
 |  lstrip(...)
 |      S.lstrip([chars]) -> str
 |      
 |      Return a copy of the string S with leading whitespace removed.
 |      If chars is given and not None, remove characters in chars instead.
 |  
 |  partition(...)
 |      S.partition(sep) -> (head, sep, tail)
 |      
 |      Search for the separator sep in S, and return the part before it,
 |      the separator itself, and the part after it.  If the separator is not
 |      found, return S and two empty strings.
 |  
 |  replace(...)
 |      S.replace(old, new[, count]) -> str
 |
 |      Return a copy of S with all occurrences of substring
 |      old replaced by new.  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(...)
 |      S.rjust(width[, fillchar]) -> str
 |
 |      Return S right-justified in a string of length width. Padding is
 |      done using the specified fill character (default is a space).
 |
 |  rpartition(...)
 |      S.rpartition(sep) -> (head, sep, tail)
 |
 |      Search for the separator sep in S, starting at the end of S, and return
 |      the part before it, the separator itself, and the part after it.  If the
 |      separator is not found, return two empty strings and S.
 |
 |  rsplit(...)
 |      S.rsplit(sep=None, maxsplit=-1) -> list of strings
 |
 |      Return a list of the words in S, using sep as the
 |      delimiter string, starting at the end of the string and
 |      working to the front.  If maxsplit is given, at most maxsplit
 |      splits are done. If sep is not specified, any whitespace string
 |      is a separator.
 |
 |  rstrip(...)
 |      S.rstrip([chars]) -> str
 |
 |      Return a copy of the string S with trailing whitespace removed.
 |      If chars is given and not None, remove characters in chars instead.
 |
 |  split(...)
 |      S.split(sep=None, maxsplit=-1) -> list of strings
 |
 |      Return a list of the words in S, using sep as the
 |      delimiter string.  If maxsplit is given, at most maxsplit
 |      splits are done. If sep is not specified or is None, any
 |      whitespace string is a separator and empty strings are
 |      removed from the result.
 |
 |  splitlines(...)
 |      S.splitlines([keepends]) -> list of strings
 |
 |      Return a list of the lines in S, 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(...)
 |      S.strip([chars]) -> str
 |
 |      Return a copy of the string S with leading and trailing
 |      whitespace removed.
 |      If chars is given and not None, remove characters in chars instead.
 |
 |  swapcase(...)
 |      S.swapcase() -> str
 |
 |      Return a copy of S with uppercase characters converted to lowercase
 |      and vice versa.
 |
 |  title(...)
 |      S.title() -> str
 |
 |      Return a titlecased version of S, i.e. words start with title case
 |      characters, all remaining cased characters have lower case.
 |
 |  translate(...)
 |      S.translate(table) -> str
 |
 |      Return a copy of the string S in which each character has been mapped
 |      through the given translation table. The table must implement
 |      lookup/indexing via __getitem__, for instance a dictionary or list,
 |      mapping Unicode ordinals to Unicode ordinals, strings, or None. If
 |      this operation raises LookupError, the character is left untouched.
 |      Characters mapped to None are deleted.
 |
 |  upper(...)
 |      S.upper() -> str
 |
 |      Return a copy of S converted to uppercase.
 |
 |  zfill(...)
 |      S.zfill(width) -> str
 |
 |      Pad a numeric string S with zeros on the left, to fill a field
 |      of the specified width. The string S is never truncated.
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  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.
```

```
help(str.startswith)
```

```
Help on method_descriptor:

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.
```

```
str.startswith?
```

```
Docstring:
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.
Type:      method_descriptor
```

```
def get_commission_amount(sales_amount, commission_pct):
    """Function to compute commission amount. commission_pct should be passed as percent
notation (eg: 20%)
       20% using percent notation is equal to 0.20 in decimal notation.
    """
    commission_amount = (sales_amount * commission_pct / 100) if commission_pct else 0
    return commission_amount
```

```
get_commission_amount
```

```
<function __main__.get_commission_amount(sales_amount, commission_pct)>
```

```
help(get_commission_amount)
```

```
Help on function get_commission_amount in module __main__:

get_commission_amount(sales_amount, commission_pct)
    Function to compute commission amount. commission_pct should be passed as percent
notation (eg: 20%)
    20% using percent notation is equal to 0.20 in decimal notation.
```

```
get_commission_amount?
```

```
Signature: get_commission_amount(sales_amount, commission_pct)
Docstring:
Function to compute commission amount. commission_pct should be passed as percent notation
(eg: 20%)
20% using percent notation is equal to 0.20 in decimal notation.
File:      ~/itversity-material/mastering-python/08_user_defined_functions/<ipython-input-5-
4e22fd73fac9>
Type:      function
```

By Durga Gadiraju
© Copyright ITVersity, Inc.