python_essentials

February 28, 2024

1 Course: Python Essentials for Data Scientists

https://courses.dataschool.io/view/courses/python-essentials-for-data-scientists

2 Basic Data Types

[1]:
```
# assignment statement
x = 4
```

[2]:
```
# change the value of x
x = 5
```

[3]:
```
# inner function runs first, outer function runs second
print(type(x))
```

```
<class 'int'>
```

[4]:
```
# return value prints out automatically
type(x)
```

[4]:
```
int
```

[5]:
```
# integer
type(5)
```

[5]:
```
int
```

[6]:
```
# floating point
type(5.0)
```

[6]:
```
float
```

[7]:
```
# string
type('five')
```

[7]:
```
str
```

[8]:
```
# string
type("five")
```

[8]:
```
str
```

[9]:
```
# use double quotes outside in order to use single quotes inside
type("we've been here")
```

[9]:
```
str
```

[10]:
```
# boolean
type(True)
```

[10]:
```
bool
```

[11]:
```
# boolean
type(False)
```

[11]:
```
bool
```

3 Lists

[12]:
```
# create a list
nums = [5, 5.0, 'five']
```

[13]:
```
# prints out the same way you input it
nums
```

[13]:
```
[5, 5.0, 'five']
```

[14]:
```
# check the type
type(nums)
```

[14]:
```
list
```

[15]:
```
# count the number of elements
len(nums)
```

[15]:
```
3
```

[16]:
```
# get help on a function
help(len)
```

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

len(obj, /)
    Return the number of items in a container.
```

[17]:
```
# get the first element
nums[0]
```

[17]:
```
5
```

[18]:
```
# change the first element
nums[0] = 6
```

[19]:
```
# lists are mutable
nums
```

[19]:
```
[6, 5.0, 'five']
```

[20]:
```
# check the type of the third element
type(nums[2])
```

[20]:
```
str
```

[21]:
```
# check the length of the third element
len(nums[2])
```

[21]:
```
4
```

[22]:
```
# append is a list method
nums.append(7)
```

[23]:
```
# list was modified without an assignment statement
nums
```

[23]:
```
[6, 5.0, 'five', 7]
```

[24]:
```
# get help on a list method
help(list.append)
```

```
Help on method_descriptor:

append(self, object, /)
    Append object to the end of the list.
```

[25]:
```
# get help on all list methods
help(list)
```

```
Help on class list in module builtins:

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
```

[26]:
```
# extend is a list method
nums.extend([8, 9])
```

```
[27]: # list was modified without an assignment statement
      nums
```

```
[27]: [6, 5.0, 'five', 7, 8, 9]
```

```
[28]: # plus sign means additions for numbers
      1 + 2
```

```
[28]: 3
```

```
[29]: # plus sign means concatenation for lists
      nums + [10, 11]
```

```
[29]: [6, 5.0, 'five', 7, 8, 9, 10, 11]
```

```
[30]: # list was not modified
      nums
```

```
[30]: [6, 5.0, 'five', 7, 8, 9]
```

```
[31]: # object can be on both sides of an assignment statement
      nums = nums + [10, 11]
```

```
[32]: # list was modified through an assignment statement
      nums
```

```
[32]: [6, 5.0, 'five', 7, 8, 9, 10, 11]
```

4 Exercise: Lists

1. Create a list of the first names of your family members.
2. Print the name of the last person in the list.
3. Print the length of the name of the first person in the list.
4. Change one of the names from their real name to their nickname.
5. Append a new person to the list.
6. Change the name of the new person to lowercase using the string method 'lower'.

After each change, check that it worked.

5 Solution: Lists

```
[33]: # exercise 1
      names = ['Wesley', 'Larry', 'Wan']
```

```
[34]: # exercise 2
      names[2]
```

```
[34]: 'Wan'
```

```
[35]: # exercise 2 using negative indexing
      names[-1]
```

```
[35]: 'Wan'
```

```
[36]: # exercise 3
      len(names[0])
```

```
[36]: 6
```

```
[37]: # exercise 4
      names[0] = 'Wes'
      names
```

```
[37]: ['Wes', 'Larry', 'Wan']
```

```
[38]: # exercise 5
      names.append('Annie')
      names
```

```
[38]: ['Wes', 'Larry', 'Wan', 'Annie']
```

```
[39]: # exercise 6 using negative indexing
      names[-1] = names[-1].lower()
      names
```

```
[39]: ['Wes', 'Larry', 'Wan', 'annie']
```

6 Comparisons & Conditional Statements

```
[40]: x = 5
```

```
[41]: # comparisons return True or False
      x > 0
```

```
[41]: True
```

```
[42]: # check for equality
      x == 0
```

```
[42]: False
```

```
[43]: # you can use logical operators: and, or, not
      x > 0 or x == 0
```

```
[43]: True
```

[44]:
```
# check for inequality
x != 0
```

[44]:
```
True
```

[45]:
```
# you can chain comparisons together
0 < x < 3
```

[45]:
```
False
```

[46]:
```
# conditional statements respond to conditions
if x > 0:
    print('positive')
```

```
positive
```

[47]:
```
# first and third lines will always run, second line will run if condition is␣
 ↪True
if x > 0:
    print('positive')
print('done checking')
```

```
positive
done checking
```

[48]:
```
x = -1
```

[49]:
```
# second line doesn't run since condition is False
if x > 0:
    print('positive')
print('done checking')
```

```
done checking
```

[50]:
```
x = 5
```

[51]:
```
# vertical whitespace and within-line whitespace don't matter
if x>0:
    print('positive')

print('done checking')
```

```
positive
done checking
```

[52]:
```
# else statement shouldn't have a condition
if x >= 0:
    print('positive')
    print('or possibly zero')
else:
    print('negative')
```

```
positive
or possibly zero
```

[53]:
```
# you can include multiple elif statements
if x > 0:
    print('positive')
elif x == 0:
    print('zero')
else:
    print('negative')
```

```
positive
```

[54]:
```
# second condition wasn't checked since the first condition was True
if x > 0:
    print('positive')
elif x > 1:
    print('more than 1')
else:
    print('negative')
```

```
positive
```

7 Functions

[55]:
```
nums = [3, 5, 4]
```

[56]:
```
# return the list in ascending order
sorted(nums)
```

[56]:
```
[3, 4, 5]
```

[57]:
```
# reverse is a parameter, True is an argument
sorted(nums, reverse=True)
```

[57]:
```
[5, 4, 3]
```

[58]:
```
# key and reverse parameters have default values
help(sorted)
```

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

sorted(iterable, /, *, key=None, reverse=False)
    Return a new list containing all items from the iterable in ascending order.

    A custom key function can be supplied to customize the sort order, and the
```

reverse flag can be set to request the result in descending order.

[59]:
```
# define a function with no parameters
def give_me_five():
    return 5
```

[60]:
```
# run the function, which prints out the return value
give_me_five()
```

[60]:
```
5
```

[61]:
```
# save the return value, thus it doesn't print out
num = give_me_five()
```

[62]:
```
num
```

[62]:
```
5
```

[63]:
```
# define a function with three parameters
def calc(a, b, operation):
    if operation == 'add':
        return a + b
    elif operation == 'subtract':
        return a - b
    else:
        return None
```

[64]:
```
# run the function without including parameter names
calc(5, 3, 'add')
```

[64]:
```
8
```

[65]:
```
# run the function with parameter names
calc(a=5, b=3, operation='add')
```

[65]:
```
8
```

[66]:
```
calc(5, 3, 'subtract')
```

[66]:
```
2
```

[67]:
```
# returns None
calc(5, 3, 'multiply')
```

[68]:
```
# error since only two arguments were provided
calc(5, 3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[68], line 2
      1 # error since only two arguments were provided
----> 2 calc(5, 3)

TypeError: calc() missing 1 required positional argument: 'operation'
```

[69]:
```
# provide a default value for operation
def calc(a, b, operation='add'):
    if operation == 'add':
        return a + b
    elif operation == 'subtract':
        return a - b
    else:
        return None
```

[70]:
```
calc(5, 3, 'add')
```

[70]:
```
8
```

[71]:
```
calc(5, 3, 'subtract')
```

[71]:
```
2
```

[72]:
```
calc(5, 3, 'multiply')
```

[73]:
```
# does not error since operation has a default value
calc(5, 3)
```

[73]:
```
8
```

[74]:
```
# display the function signature
help(calc)
```

```
Help on function calc in module __main__:

calc(a, b, operation='add')
    # provide a default value for operation
```

[75]:
```
x = 8
y = 2
```

[76]:
```
# you can pass other variables into a function
calc(x, y)
```

[76]: 10

[77]:
```
# a and b only exist within the function
a
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[77], line 2
      1 # a and b only exist within the function
----> 2 a

NameError: name 'a' is not defined
```

8 Exercise: Functions

Write a function that takes two arguments (hours and hourly_rate) and returns the total pay. If the hourly rate is not specified, it should default to 100.

9 Solution: Functions

[78]:
```
# hourly_rate has a default value, but hours does not
def compute_pay(hours, hourly_rate=100):
    return hours * hourly_rate
```

[79]:
```
# positional arguments
compute_pay(30, 50)
```

[79]: 1500

[80]:
```
# keyword arguments
compute_pay(hours=30, hourly_rate=50)
```

[80]: 1500

[81]:
```
# uses default hourly_rate of 100
compute_pay(30)
```

[81]: 3000

10 Exercise: Functions & Conditional Statements

Update the function to provide 1.5 times the hourly rate for hours worked above 40 hours. Return the total pay.

(In other words: The regular hourly rate should be used for the **first 40 hours**, and 1.5 times the hourly rate should be used for **any additional hours**.)

11 Solution: Functions & Conditional Statements

```
[82]: # use parentheses within your code to control order of operations
      def compute_pay(hours, hourly_rate=100):
          if hours <= 40:
              return hours * hourly_rate
          else:
              base_pay = hourly_rate * 40
              overtime_pay = (hours - 40) * (hourly_rate * 1.5)
              return base_pay + overtime_pay
```

```
[83]: # 30 * 100
      compute_pay(30)
```

```
[83]: 3000
```

```
[84]: # 40 * 100
      compute_pay(40)
```

```
[84]: 4000
```

```
[85]: # 40 * 100 + 10 * 150
      compute_pay(50)
```

```
[85]: 5500.0
```

```
[86]: # 40 * 10 + 10 * 15
      compute_pay(50, 10)
```

```
[86]: 550.0
```

12 Naming Objects

```
[87]: # this is allowed, but isn't a good idea
      list = 5
```

```
[88]: type(list)
```

```
[88]: int
```

```
[89]: # delete an object
      del list
```

```
[90]: type(list)
```

```
[90]: type
```

13 Writing Comments

[91]:
```
def compute_pay(hours, hourly_rate=100):
    """
    Compute total pay for permanent employees

    example: xyz
    """
    if hours <= 40:
        return hours * hourly_rate
    else:
        # overtime pay is 1.5x and starts at 40 hours
        base_pay = hourly_rate * 40
        # print(base_pay)
        overtime_pay = (hours - 40) * (hourly_rate * 1.5)    # don't include␣
 ↪temp workers
        # print(overtime_pay)
        return base_pay + overtime_pay

    """
    Note: Fix this to exclude temp workers
    Overtime rate will change to 1.6x in 2020
    """

    # multi-line
    # comments
```

[92]:
```
# includes the function signature and the docstring
help(compute_pay)
```

```
Help on function compute_pay in module __main__:

compute_pay(hours, hourly_rate=100)
    Compute total pay for permanent employees

    example: xyz
```

14 List Slicing

[93]:
```
weekdays = ['mon', 'tues', 'wed', 'thurs', 'fri']
```

[94]:
```
# get the first element
weekdays[0]
```

[94]:
```
'mon'
```

[95]:
```
# get the last element
weekdays[-1]
```

[95]:
```
'fri'
```

[96]:
```
# get elements 0 through 2
weekdays[0:3]
```

[96]:
```
['mon', 'tues', 'wed']
```

[97]:
```
# get elements 0 through 2
weekdays[:3]
```

[97]:
```
['mon', 'tues', 'wed']
```

[98]:
```
# get elements 3 through 4
weekdays[3:5]
```

[98]:
```
['thurs', 'fri']
```

[99]:
```
# get element 3 through the end of the list
weekdays[3:]
```

[99]:
```
['thurs', 'fri']
```

[100]:
```
# get all elements
weekdays[:3] + weekdays[3:]
```

[100]:
```
['mon', 'tues', 'wed', 'thurs', 'fri']
```

[101]:
```
# get all elements
weekdays[:2] + weekdays[2:]
```

[101]:
```
['mon', 'tues', 'wed', 'thurs', 'fri']
```

[102]:
```
# get elements 0 through 2 with a step size of 1
weekdays[0:3:1]
```

[102]:
```
['mon', 'tues', 'wed']
```

[103]:
```
# get elements 0 through 2 with a step size of 2
weekdays[0:3:2]
```

[103]:
```
['mon', 'wed']
```

[104]:
```
# get all elements with a step size of 2
weekdays[::2]
```

[104]:
```
['mon', 'wed', 'fri']
```

[105]:
```
# traverse the list backwards
weekdays[::-1]
```

[105]:
```
['fri', 'thurs', 'wed', 'tues', 'mon']
```

[106]:
```
# save a slice of weekdays
early_week = weekdays[:3]
early_week
```

[106]:
```
['mon', 'tues', 'wed']
```

[107]:
```
# weekdays list was not affected
weekdays
```

[107]:
```
['mon', 'tues', 'wed', 'thurs', 'fri']
```

15 Strings

[108]:
```
# strings are ordered containers of characters
word = 'hello'
```

[109]:
```
# count the number of characters
len(word)
```

[109]:
```
5
```

[110]:
```
# get the first character
word[0]
```

[110]:
```
'h'
```

[111]:
```
# get characters 1 through 2
word[1:3]
```

[111]:
```
'el'
```

[112]:
```
# get the last character
word[-1]
```

[112]:
```
'o'
```

[113]:
```
# plus sign means concatenation for strings
word + ' there'
```

[113]:
```
'hello there'
```

[114]:
```
# concatenation requires the same data types
5 + ' is a number'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[114], line 2
      1 # concatenation requires the same data types
----> 2 5 + ' is a number'

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
[115]: # convert an integer to a string
       str(5)
```

```
[115]: '5'
```

```
[116]: # concatenate two strings
       str(5) + ' is a number'
```

```
[116]: '5 is a number'
```

```
[117]: # convert a string to an integer
       int('5')
```

```
[117]: 5
```

```
[118]: # error since strings are immutable
       word[0] = 'H'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[118], line 2
      1 # error since strings are immutable
----> 2 word[0] = 'H'

TypeError: 'str' object does not support item assignment
```

```
[119]: # return a titlecased version of the string
       word.title()
```

```
[119]: 'Hello'
```

```
[120]: # get help on a string method
       help(str.title)
```

```
Help on method_descriptor:

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.
```

[121]:
```
# get help on all string methods
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.
 |
 |      Call keyword.iskeyword(s) to test whether string s is a reserved
identifier,
 |      such as "def" or "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.
 |
 |  removeprefix(self, prefix, /)
 |      Return a str with the given prefix string removed if present.
 |
 |      If the string starts with the prefix string, return
```

```
string[len(prefix):].
 |      Otherwise, return a copy of the original string.
 |
 |  removesuffix(self, suffix, /)
 |      Return a str with the given suffix string removed if present.
 |
 |      If the string ends with the suffix string and that suffix is not empty,
 |      return string[:-len(suffix)]. Otherwise, return a copy of the original
 |      string.
 |
 |  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 substrings in the string, using sep as the
separator string.
 |
 |        sep
 |          The separator used to split the string.
 |
 |          When set to None (the default value), will split on any whitespace
 |          character (including \\n \\r \\t \\f and spaces) and will discard
 |          empty strings from the result.
 |        maxsplit
 |          Maximum number of splits (starting from the left).
 |          -1 (the default value) means no limit.
 |
 |      Splitting starts at the end of the string and works 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 substrings in the string, using sep as the
separator string.
 |
 |        sep
 |          The separator used to split the string.
 |
 |          When set to None (the default value), will split on any whitespace
 |          character (including \\n \\r \\t \\f and spaces) and will discard
 |          empty strings from the result.
 |        maxsplit
 |          Maximum number of splits (starting from the left).
 |          -1 (the default value) means no limit.
 |
 |      Note, str.split() is mainly useful for data that has been intentionally
 |      delimited.  With natural text that includes punctuation, consider using
 |      the regular expression module.
 |
 |  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(…)
 |      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.
```

[122]:
```
# error since lists don't have a title method
weekdays.title()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[122], line 2
      1 # error since lists don't have a title method
----> 2 weekdays.title()

AttributeError: 'list' object has no attribute 'title'
```

[123]:
```
# error since strings don't have an append method
word.append('a')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[123], line 2
      1 # error since strings don't have an append method
----> 2 word.append('a')

AttributeError: 'str' object has no attribute 'append'
```

16 Exercise: Strings & Slicing

Write a function that accepts a string and returns a modified string:

- If the original string is at least 4 characters long, return a string made of its first 2 and last 2 characters.
- Otherwise, return an empty string.

Example:

- 'python' returns 'pyon'
- 'list' returns 'list'
- 'abc' returns ' '

17 Solution: Strings & Slicing

[124]:
```
def string_slicer(string):
    if len(string) >= 4:
        return string[:2] + string[-2:]
    else:
        return ''
```

[125]:
```
string_slicer('python')
```

[125]:
```
'pyon'
```

[126]:
```
string_slicer('list')
```

[126]:
```
'list'
```

[127]:
```
string_slicer('abc')
```

[127]:
```
''
```

18 For Loops

[128]:
```
nums = [1, 2, 3, 4, 5]
```

[129]:
```
# iterate through nums, each integer becomes num one time
for num in nums:
    print(num)
```

```
1
2
3
4
5
```

[130]:
```python
# name of the temporary variable doesn't matter
for x in nums:
    print(x)
```

```
1
2
3
4
5
```

[131]:
```python
word = 'hello'
```

[132]:
```python
# iterate through word, each character becomes letter one time
for letter in word:
    print(letter)
```

```
h
e
l
l
o
```

[133]:
```python
# inappropriate variable names make your code less readable
for num in word:
    print(num)
```

```
h
e
l
l
o
```

[134]:
```python
# don't access the object you are iterating through within the loop
for letter in word:
    print(word)
```

```
hello
hello
hello
hello
hello
```

[135]:
```python
# for loop body runs 5 times, last line runs 1 time
for letter in word:
    print(letter)
    print('end of loop')
print('complete')
```

```
h
end of loop
```

```
e
end of loop
l
end of loop
l
end of loop
o
end of loop
complete
```

19 Exercise: For Loops

Create a list called ‘fruits’ that contains ‘apple’, ‘banana’, and ‘cherry’. Then, write a for loop that prints:

```
APPLE
BANANA
CHERRY
```

20 Solution: For Loops

[136]:
```python
fruits = ['apple', 'banana', 'cherry']
```

[137]:
```python
# fruit is a string, upper is a string method
for fruit in fruits:
    print(fruit.upper())
```

```
APPLE
BANANA
CHERRY
```

[138]:
```python
# comment out code to preserve it without running it
# help(str)
```

[139]:
```python
# you can include conditional statements within a for loop
for fruit in fruits:
    if len(fruit) > 5:
        print(fruit.upper())
```

```
BANANA
CHERRY
```

21 List Comprehensions

[140]:
```python
nums
```

[140]:
```
[1, 2, 3, 4, 5]
```

```
[141]: for num in nums:
           print(num * 2)
```

```
2
4
6
8
10
```

```
[142]: # store the results in a list instead of printing them
       doubled = []
       for num in nums:
           doubled.append(num * 2)
```

```
[143]: doubled
```

```
[143]: [2, 4, 6, 8, 10]
```

```
[144]: # equivalent list comprehension
       doubled = [num * 2 for num in nums]
```

```
[145]: doubled
```

```
[145]: [2, 4, 6, 8, 10]
```

```
[146]: # resulting list does not need to be assigned a name
       [num * 2 for num in nums]
```

```
[146]: [2, 4, 6, 8, 10]
```

22 Exercise: List Comprehensions

Create a list called 'fruits' that contains 'apple', 'banana', and 'cherry'.

1. Write a list comprehension that returns `['APPLE', 'BANANA', 'CHERRY']`
2. Write a list comprehension that returns `['a', 'b', 'c']`

23 Solution: List Comprehensions

```
[147]: fruits = ['apple', 'banana', 'cherry']
```

```
[148]: # rewrite this for loop for exercise 1
       for fruit in fruits:
           print(fruit.upper())
```

```
APPLE
BANANA
CHERRY
```

```
[149]: # exercise 1: use the upper method on each fruit
       [fruit.upper() for fruit in fruits]

[149]: ['APPLE', 'BANANA', 'CHERRY']

[150]: # exercise 2: get the first letter from each fruit
       [fruit[0] for fruit in fruits]

[150]: ['a', 'b', 'c']
```

24 Dictionaries

```
[151]: # dictionaries contain key-value pairs
       family = {'dad':'Homer', 'mom':'Marge', 'size':2}

[152]: # keys are separated from values by a colon
       family

[152]: {'dad': 'Homer', 'mom': 'Marge', 'size': 2}

[153]: # count the number of key-value pairs
       len(family)

[153]: 3

[154]: # use a key to look up a value
       family['dad']

[154]: 'Homer'

[155]: # you can't use a value to look up a key
       family['Homer']

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[155], line 2
      1 # you can't use a value to look up a key
----> 2 family['Homer']

KeyError: 'Homer'

[156]: # add a new key-value pair to the dictionary
       family['cat'] = 'Snowball'

[157]: # dictionaries are ordered as of Python 3.7
       family
```

```
[157]: {'dad': 'Homer', 'mom': 'Marge', 'size': 2, 'cat': 'Snowball'}
```

```
[158]: # you can't look up values by position
       family[0]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[158], line 2
      1 # you can't look up values by position
----> 2 family[0]

KeyError: 0
```

```
[159]: # keys must be unique, so this overwrites the original value
       family['cat'] = 'Snowball II'
```

```
[160]: family
```

```
[160]: {'dad': 'Homer', 'mom': 'Marge', 'size': 2, 'cat': 'Snowball II'}
```

```
[161]: # delete a key-value pair
       del family['cat']
```

```
[162]: family
```

```
[162]: {'dad': 'Homer', 'mom': 'Marge', 'size': 2}
```

25 Lists vs. Dictionaries

lists	dictionaries
contain elements	contain key-value pairs
accessed by position	accessed by key
can contain duplicates	keys must be unique
use brackets	use braces
designed for flexibility	designed for performance

26 Exercise: Dictionaries

1. Create a dictionary of your own family (called 'my_family'), similar to the Simpsons dictionary.
2. Check the dictionary length.
3. Use a key to look up a value.
4. Add a new person to the family.
5. Change the value of 'size' to reflect the new person.

27 Solution: Dictionaries

```
[163]: # exercise 1
       my_family = {'dad':'Larry', 'mom':'Wan', 'brother':'Wes', 'size':4}
```

```
[164]: # exercise 2
       len(my_family)
```

```
[164]: 4
```

```
[165]: # exercise 3
       my_family['dad']
```

```
[165]: 'Larry'
```

```
[166]: # exercise 4
       my_family['wife'] = 'Annie'
       my_family
```

```
[166]: {'dad': 'Larry', 'mom': 'Wan', 'brother': 'Wes', 'size': 4, 'wife': 'Annie'}
```

```
[167]: # exercise 5
       my_family['size'] = 5
       my_family
```

```
[167]: {'dad': 'Larry', 'mom': 'Wan', 'brother': 'Wes', 'size': 5, 'wife': 'Annie'}
```

28 Nested Data

```
[168]: family
```

```
[168]: {'dad': 'Homer', 'mom': 'Marge', 'size': 2}
```

```
[169]: # add a new key-value pair
       family['kids'] = ['bart', 'lisa']
```

```
[170]: # dictionary value can be any data type, including a list
       family
```

```
[170]: {'dad': 'Homer', 'mom': 'Marge', 'size': 2, 'kids': ['bart', 'lisa']}
```

```
[171]: # count the number of key-value pairs
       len(family)
```

```
[171]: 4
```

```
[172]: # value is a list
       family['kids']
```

```
[172]: ['bart', 'lisa']
```

```
[173]: # get the first element of the list
       family['kids'][0]
```

```
[173]: 'bart'
```

```
[174]: type(family['kids'])
```

```
[174]: list
```

```
[175]: type(family['kids'][0])
```

```
[175]: str
```

29 Exercise: Nested Data

Before starting this exercise, the 'family' dictionary should contain the following:

```
[176]: family
```

```
[176]: {'dad': 'Homer', 'mom': 'Marge', 'size': 2, 'kids': ['bart', 'lisa']}
```

1. Get the name of the girl from the kids list.
2. Print the length of the kids list.
3. Add 'Maggie' to the kids list.
4. Fix 'bart' and 'lisa' so that the first letter of each is capitalized (using one or more assignment statements).
5. Now change all three kids' names to UPPERCASE using a for loop or list comprehension.

30 Solution: Nested Data

```
[177]: # exercise 1
       family['kids'][1]
```

```
[177]: 'lisa'
```

```
[178]: # exercise 2
       len(family['kids'])
```

```
[178]: 2
```

```
[179]: # exercise 3
       family['kids'].append('Maggie')
       family
```

```
[179]: {'dad': 'Homer', 'mom': 'Marge', 'size': 2, 'kids': ['bart', 'lisa', 'Maggie']}
```

[180]:
```
# exercise 4
family['kids'][0] = 'Bart'
family['kids'][1] = family['kids'][1].title()
family
```

[180]:
```
{'dad': 'Homer', 'mom': 'Marge', 'size': 2, 'kids': ['Bart', 'Lisa', 'Maggie']}
```

[181]:
```
# exercise 5 using a list comprehension
family['kids'] = [kid.upper() for kid in family['kids']]
family
```

[181]:
```
{'dad': 'Homer', 'mom': 'Marge', 'size': 2, 'kids': ['BART', 'LISA', 'MAGGIE']}
```

[182]:
```
# exercise 5 using a for loop
kids_uppercase = []
for kid in family['kids']:
    kids_uppercase.append(kid.upper())
family['kids'] = kids_uppercase
family
```

[182]:
```
{'dad': 'Homer', 'mom': 'Marge', 'size': 2, 'kids': ['BART', 'LISA', 'MAGGIE']}
```

31 Nested Data & Tuples

[183]:
```
family
```

[183]:
```
{'dad': 'Homer', 'mom': 'Marge', 'size': 2, 'kids': ['BART', 'LISA', 'MAGGIE']}
```

[184]:
```
# return a list-like object containing the keys
family.keys()
```

[184]:
```
dict_keys(['dad', 'mom', 'size', 'kids'])
```

[185]:
```
# return a list-like object containing the values
family.values()
```

[185]:
```
dict_values(['Homer', 'Marge', 2, ['BART', 'LISA', 'MAGGIE']])
```

[186]:
```
# convert it to a list
vals = list(family.values())
```

[187]:
```
# list within a list
vals
```

[187]:
```
['Homer', 'Marge', 2, ['BART', 'LISA', 'MAGGIE']]
```

```
[188]: # get the inner list
       vals[-1]
```

```
[188]: ['BART', 'LISA', 'MAGGIE']
```

```
[189]: # get the first element of the inner list
       vals[-1][0]
```

```
[189]: 'BART'
```

```
[190]: # return a list-like object containing the keys and values
       family.items()
```

```
[190]: dict_items([('dad', 'Homer'), ('mom', 'Marge'), ('size', 2), ('kids', ['BART',
       'LISA', 'MAGGIE'])])
```

```
[191]: # convert it to a list
       items = list(family.items())
```

```
[192]: # list of 4 tuples
       items
```

```
[192]: [('dad', 'Homer'),
        ('mom', 'Marge'),
        ('size', 2),
        ('kids', ['BART', 'LISA', 'MAGGIE'])]
```

```
[193]: # get the first tuple
       items[0]
```

```
[193]: ('dad', 'Homer')
```

```
[194]: # get the second tuple
       items[1]
```

```
[194]: ('mom', 'Marge')
```

```
[195]: # get the first element of the tuple
       items[1][0]
```

```
[195]: 'mom'
```

```
[196]: # error since tuples are immutable
       items[1][0] = 'MOM'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[196], line 2
```

```
      1 # error since tuples are immutable
----> 2 items[1][0] = 'MOM'

TypeError: 'tuple' object does not support item assignment
```

32 Exercise: Nested Data & Tuples (Part 1)

Here is a data structure:

```
[197]: vals
```

```
[197]: ['Homer', 'Marge', 2, ['BART', 'LISA', 'MAGGIE']]
```

Without running the code, figure out what each of these functions would return:

1. `len(vals)`
2. `len(vals[3])`
3. `len(vals[1])`

33 Solution: Nested Data & Tuples (Part 1)

```
[198]: # exercise 1: length of the outer list
       len(vals)
```

```
[198]: 4
```

```
[199]: # exercise 2: length of the inner list
       len(vals[3])
```

```
[199]: 3
```

```
[200]: # exercise 3: length of 'Marge'
       len(vals[1])
```

```
[200]: 5
```

```
[201]: vals[1]
```

```
[201]: 'Marge'
```

34 Exercise: Nested Data & Tuples (Part 2)

Here is another data structure:

```
[202]: items
```

```
[202]: [('dad', 'Homer'),
        ('mom', 'Marge'),
        ('size', 2),
        ('kids', ['BART', 'LISA', 'MAGGIE'])]
```

Without running the code, figure out what each of these functions would return:

1. `len(items[0])`
2. `len(items[-1][-1][-1])`

35 Solution: Nested Data & Tuples (Part 2)

```
[203]: # exercise 1: length of the first tuple
       len(items[0])
```

```
[203]: 2
```

```
[204]: items[0]
```

```
[204]: ('dad', 'Homer')
```

```
[205]: # exercise 2: length of 'MAGGIE'
       len(items[-1][-1][-1])
```

```
[205]: 6
```

```
[206]: items[-1]
```

```
[206]: ('kids', ['BART', 'LISA', 'MAGGIE'])
```

```
[207]: items[-1][-1]
```

```
[207]: ['BART', 'LISA', 'MAGGIE']
```

```
[208]: items[-1][-1][-1]
```

```
[208]: 'MAGGIE'
```

```
[209]: items[-1][-1][-1][-1]
```

```
[209]: 'E'
```

36 Imports

```
[210]: # error since sqrt function has not been imported
       sqrt(49)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[210], line 2
      1 # error since sqrt function has not been imported
----> 2 sqrt(49)

NameError: name 'sqrt' is not defined
```

[211]:
```
# import a module from the Python standard library
import math
```

[212]:
```
# use the sqrt function from the math module
math.sqrt(49)
```

[212]:
```
7.0
```

[213]:
```
# append is an available method because nums is a list
nums.append(6)
```

[214]:
```
# define an alias for math
import math as xyz
```

[215]:
```
# use the sqrt function from the math module via an alias
xyz.sqrt(49)
```

[215]:
```
7.0
```

[216]:
```
# import a single function from the math module
from math import sqrt
```

[217]:
```
# use the sqrt function without typing the module name
sqrt(49)
```

[217]:
```
7.0
```