1 - Python Basics

March 22, 2023

Table of Contents

1 Simple Data Types

1.1 Numbers

Numbers are: int, float, complex

1.1.1 Integers

[1]:
```
5+5
```

[1]:
```
10
```

[2]:
```
5-7
```

[2]:
```
-2
```

[3]:
```
5*5
```

[3]:
```
25
```

[4]:
```
# Simple division

25/3
```

[4]:
```
8.333333333333334
```

1.1.2 Floating points (aka numbers with decimals)

[5]:
```
a = 50 * 23/4
a
```

[5]: 287.5

1.1.3 A complex number

[6]:
```
a = 23e4
b = 23E4
c = -23e100

print(a)
print(b)
print("This is C: {}".format(c))
```

```
230000.0
230000.0
This is C: -2.3e+101
```

1.2 Arithmetic Operators

- Addition
- Subtraction
- Multiplication
- Division
- Modulus (remainder)
- Exponents
- Floor division

[7]:
```
# Modulus - remainder of division

25 % 3
```

[7]: 1

[8]:
```
# Calculate power

5**2  # squared

# 5**5 = 3125. 5 to the 5th power
```

[8]: 25

[9]:
```
5**5
```

[9]: 3125

[10]:
```
# Floor division

25 // 3
```

[10]: 8

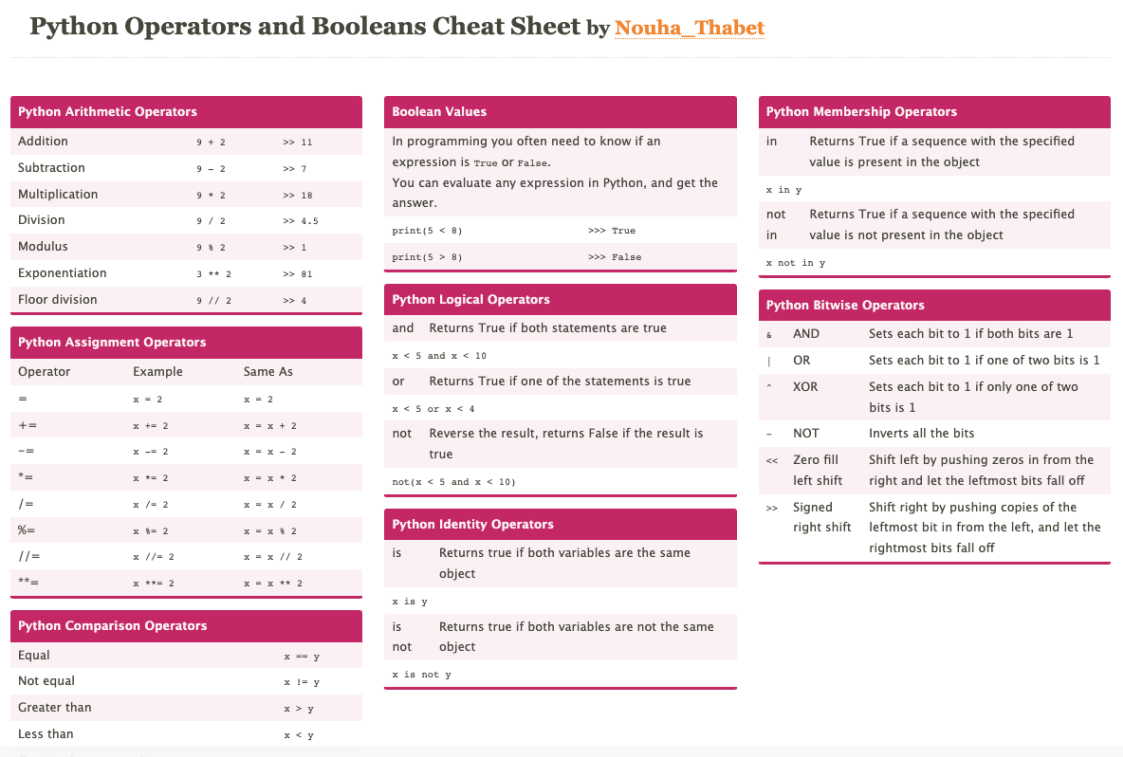
1.2.1 Assign a value to a variable

[11]:
```
shipments = 1000
administered = 950
doses_left = shipments - administered
doses_left
```

[11]: 50

1.3 Operators Cheat Sheet

https://cheatography.com/nouha-thabet/cheat-sheets/python-operators-and-booleans/



Python Operators and Booleans Cheat Sheet by Nouha_Thabet

Python Arithmetic Operators

Addition	`9 + 2`	`>> 11`
Subtraction	`9 - 2`	`>> 7`
Multiplication	`9 * 2`	`>> 18`
Division	`9 / 2`	`>> 4.5`
Modulus	`9 % 2`	`>> 1`
Exponentiation	`3 ** 2`	`>> 81`
Floor division	`9 // 2`	`>> 4`

Python Assignment Operators

Operator	Example	Same As
=	`x = 2`	`x = 2`
+=	`x += 2`	`x = x + 2`
-=	`x -= 2`	`x = x - 2`
*=	`x *= 2`	`x = x * 2`
/=	`x /= 2`	`x = x / 2`
%=	`x %= 2`	`x = x % 2`
//=	`x //= 2`	`x = x // 2`
**=	`x **= 2`	`x = x ** 2`

Python Comparison Operators

Equal	`x == y`
Not equal	`x != y`
Greater than	`x > y`
Less than	`x < y`

Boolean Values

In programming you often need to know if an expression is `True` or `False`.
You can evaluate any expression in Python, and get the answer.

`print(5 < 8)` `>>> True`

`print(5 > 8)` `>>> False`

Python Logical Operators

and	Returns True if both statements are true

`x < 5 and x < 10`

or	Returns True if one of the statements is true

`x < 5 or x < 4`

not	Reverse the result, returns False if the result is true

`not(x < 5 and x < 10)`

Python Identity Operators

is	Returns true if both variables are the same object

`x is y`

is not	Returns true if both variables are not the same object

`x is not y`

Python Membership Operators

in	Returns True if a sequence with the specified value is present in the object

`x in y`

not in	Returns True if a sequence with the specified value is not present in the object

`x not in y`

Python Bitwise Operators

&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

[12]:
```
'yes' == 'no'
```

[12]: False

[13]:
```
(10>12) and (30<40)
```

[13]:
```
False
```

[14]:
```
(10>12) or (30<40)
```

[14]:
```
True
```

[15]:
```
(10==12) or (1==1) or (30>40)
```

[15]:
```
True
```

[16]:
```
(10==12) or (1==1) and (30>40)
```

[16]:
```
False
```

1.4 Strings

[17]:
```
'use a single quote'
```

[17]:
```
'use a single quote'
```

[18]:
```
"use a double quote"
```

[18]:
```
'use a double quote'
```

[19]:
```
"'this is quote' used in a quote"
```

[19]:
```
"'this is quote' used in a quote"
```

[20]:
```
'"this is a quote" in a quote'
```

[20]:
```
'"this is a quote" in a quote'
```

[21]:
```
2*'su- '+'sudio.  Thank you Phil Collins'
```

[21]:
```
'su- su- sudio.  Thank you Phil Collins'
```

1.4.1 Working with a long string

[22]:
```
lyrics = ("There's this girl that's been on my mind "
          'All the time, Su-Sussudio '
          'Oh oh '
          "Now she don't even know my name "
          'But I think she likes me just the same '
          'Su-Sussudio '
          'Woah oh ')
lyrics
```

[22]:
```
"There's this girl that's been on my mind All the time, Su-Sussudio Oh oh Now
she don't even know my name But I think she likes me just the same Su-Sussudio
Woah oh "
```

1.4.2 Reference Material

- https://docs.python.org/3/library/stdtypes.html#string-methods or
- https://www.w3schools.com/python/python_strings_methods.asp

1.5 Booleans

A boolean can have 1 of 2 values: True or False.

[23]:
```
x = 100< 95
x
```

[23]:
```
False
```

[24]:
```
a = 100
b = 95
x = a > b
if x == True: print('yes')
```

```
yes
```

[25]:
```
type(a)
```

[25]:
```
int
```

2 Data Structures

- List
- Tuple
- Set
- Dictionary

https://docs.python.org/3/tutorial/datastructures.html

2.1 Lists

Lists, Tuples, Sets, Dictionaries

A compound data type

- items, sepated by commas
- inside square brackets []
- usually, but not always the same data type

[26]:
```
months = [1,2,3,4,5,6,7,8,9,10,11,12]
months
```

[26]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

[27]:
```
month_names = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',␣
↪'Oct', 'Nov', 'Dec']
month_names
```

[27]:
```
['Jan',
 'Feb',
 'Mar',
 'Apr',
 'May',
 'Jun',
 'Jul',
 'Aug',
 'Sep',
 'Oct',
 'Nov',
 'Dec']
```

[28]:
```
letters = ['a', 'b', 'c', 'd']
letters
```

[28]:
```
['a', 'b', 'c', 'd']
```

[29]:
```
mess = letters+months
mess
```

[29]:
```
['a', 'b', 'c', 'd', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

2.1.1 Indexing and Slicing

Indexing returns a value from a list based on its position. **Indexing starts at zero** The index values(s) go inside square brackets []. **Slicing** returns a new list.

[30]:
```
# Indexing
# Return the value of letters at index 0.
letters = ['a', 'b', 'c', 'd']
letters[0]
```

[30]:
```
'a'
```

[31]:
```
# Return 'c'
letters[2]
```

[31]:
```
'c'
```

[32]:
```
# Work from the end back.  This do NOT start at zero
# Return 'c' starting from the end.

letters[-2]
```

[32]:
```
'c'
```

[33]:
```
# Slicing - returns a new list based on the index provided
# Return 6, 7 and 8

month_names[6:9]
```

[33]:
```
['Jul', 'Aug', 'Sep']
```

[34]:
```
# Return everything from 6 on
```

[35]:
```
month_names[6:]
```

[35]:
```
['Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
```

2.1.2 Replace a value

[36]:
```
letters[2]='Z'
letters
```

[36]:
```
['a', 'b', 'Z', 'd']
```

2.1.3 Add values

- append
- insert
- extend

[37]:
```
# Using the string function APPEND.
letters = ['a', 'b', 'c', 'd']

letters.append('e')  # .append takes one parameter
letters
```

[37]:
```
['a', 'b', 'c', 'd', 'e']
```

[38]:
```
# Using the string function INSERT to put something in at a specific position.
letters = ['a', 'b', 'c', 'd']

letters.insert(2,'ZZ')
```

```
letters
```

[38]: ['a', 'b', 'ZZ', 'c', 'd']

[39]:
```
# Using EXTEND to make the list bigger
letters = ['a', 'b', 'c', 'd']

abc = ['f','g','h']
letters.extend(abc)
letters
```

[39]: ['a', 'b', 'c', 'd', 'f', 'g', 'h']

[40]:
```
# This is what happens if we try APPEND.

abc = ['f','g','h']
letters.append(abc)  # Returns a nested list
letters
```

[40]: ['a', 'b', 'c', 'd', 'f', 'g', 'h', ['f', 'g', 'h']]

[41]:
```
# So, how can we make a big list with letters and abc?
letters = ['a', 'b', 'c', 'd']
letters+abc
```

[41]: ['a', 'b', 'c', 'd', 'f', 'g', 'h']

2.1.4 Remove values

- pop
- remove
- clear

[42]:
```
letters = ['a', 'b', 'c', 'd']

letters.pop(1)
letters
```

[42]: ['a', 'c', 'd']

[43]:
```
# POP without an index value removes the last value.

letters = ['a', 'b', 'c', 'd']
letters.pop()
letters
```

[43]: ['a', 'b', 'c']

[44]:
```
# REMOVE
letters = ['a', 'b', 'c', 'd', 'a']

letters.remove('a')
letters
```

[44]:
```
['b', 'c', 'd', 'a']
```

[45]:
```
# CLEAR the list
letters = ['a', 'b', 'c', 'd']

letters.clear()
letters
```

[45]:
```
[]
```

2.1.5 Sort, Count, Reverse

[46]:
```
# Sort the list
letters = ['a', 'b', 'c', 'd','a','a']

letters.sort()
letters

# letters.sort(reverse=True)
# letters
```

[46]:
```
['a', 'a', 'a', 'b', 'c', 'd']
```

[47]:
```
letters = ['a', 'b', 'c', 'd','a','a']

letters.reverse()
letters
```

[47]:
```
['a', 'a', 'd', 'c', 'b', 'a']
```

[48]:
```
letters = ['a', 'b', 'c', 'd','a','a']

letters.count('a')
```

[48]:
```
3
```

2.1.6 Miscellaneous

List can be used as:

- **stacks** for last-in, first-out processing
- **queues** for first-in, first-out processing

2.2 Tuples

Lists, **Tuples**, Sets, Dictionaries

- Like a list, they contain a collection of items.
- A list is defined by the use of square brackets. A tuple is defined by the use of parentheses.
- Unlike a list, which can have its items changed (mutable), tuples cannot (**imutable**)
- Tuples are more memory efficient and are generally used for a set of unchanging values (e.g., months, states, etc.)

[49]:
```
letters_t = ('a', 'b', 'c', 'd')
type(letters_t)
```

[49]:
```
tuple
```

[50]:
```
import sys

letters = ['a', 'b', 'c', 'd','a','a','a', 'b', 'c', 'd','a','a','a', 'b', 'c',␣
 ↪'d','a','a']
letters_t = ('a', 'b', 'c', 'd','a','a','a', 'b', 'c', 'd','a','a','a', 'b',␣
 ↪'c', 'd','a','a')
print(sys.getsizeof(letters))
print(sys.getsizeof(letters_t))
```

```
216
184
```

2.3 Sets

Lists, Tuples, **Sets**, Dictionaries

- Like a list and a tuple, they contain a collection of items.
- A set is defined by the use of curly braces {}.
- A set is an unordered collection with **no duplicate items**.
- A set is immutable after it has been created.
- Sets are useful for removing duplicate values
- Set items are unordered (and may appear in a different order each time the set is used).

[51]:
```
letters_s = {'a', 'b', 'e','c', 'd','a','a'}
letters_s
```

[51]:
```
{'a', 'b', 'c', 'd', 'e'}
```

[52]:
```
# Items already in a set cannot be changed but new items can be added.
letters_s = {'a', 'b', 'e','c', 'd','a','a'}

letters_s.add('ZZZ')
letters_s
```

[52]:
```
{'ZZZ', 'a', 'b', 'c', 'd', 'e'}
```

[53]:
```
# Sets can be added to one another

letters_s = {'a', 'b', 'c', 'd'}
abc_s = {'f','g','h'}

letters_s.update(abc_s)
letters_s
```

[53]:
```
{'a', 'b', 'c', 'd', 'f', 'g', 'h'}
```

[54]:
```
# Items can be removed from a set.
letters_s = {'a', 'b', 'c', 'd'}

letters_s.remove('a')
letters_s

#letters_s.discard('a')
#letters_s
```

[54]:
```
{'b', 'c', 'd'}
```

[55]:
```
# Try to remove 'e' from the list.  Do you get an error message?
# Try to discard 'e'.  What happens?

#letters_s.remove('e')
#letters_s

#letters_s.discard('e')
#letters_s
```

2.4 Set Exercise - 5 minutes

Use the documentation - https://docs.python.org/3/library/stdtypes.html#set or - https://www.w3schools.com/python/python_sets_methods.asp

In the two sets below: - What items are in common (intersection)? - What items are unique across both sets (i.e., not part of the intersection?

deliveries = {‘Boston’,‘Monticello’,‘Chicago’,‘Atlanta’,‘Dickey’,‘Douglas’,‘Zenda’,‘Springfield’} clinics = {‘Boston’,‘Chicago’,‘Atlanta’,‘Zenda’,‘Springfield’}

[56]:
```
deliveries =␣
  ↪{'Boston','Monticello','Chicago','Atlanta','Dickey','Douglas','Zenda','Springfield'}
clinics = {'Boston','Chicago','Atlanta','Zenda','Springfield'}
```

2.5 Dictionaries

Lists, Tuples, Sets, **Dictionaries**

- A dictionary (aka dict) is a collection of key:value pairs (e.g., zipcode:01545)
- A dictionary is defined by the use of curly braces {} that contain key:value pairs.
- A dictionary is an ordered collection with **no duplicate keys**.
- A dictionary is mutable.

[57]:
```
things = {'key1':'thing1','key2':'thing2','key3':'thing3'}
type(things)
```

[57]:
```
dict
```

[58]:
```
# Get the value of a key

things.get('key1')
```

[58]:
```
'thing1'
```

[59]:
```
# Get the keys

things.keys()
```

[59]:
```
dict_keys(['key1', 'key2', 'key3'])
```

[60]:
```
# Add a key:value pair to the dictionary

things['key4']='thing4'
things
```

[60]:
```
{'key1': 'thing1', 'key2': 'thing2', 'key3': 'thing3', 'key4': 'thing4'}
```

[61]:
```
my_dict = {}
my_dict['l'] = 4
my_dict['m'] = 6
my_dict['n'] = 9
new_value = list(my_dict.items())  # Cast the dict to a list in order to access␣
 ↪by index

new_value
```

[61]:
```
[('l', 4), ('m', 6), ('n', 9)]
```

[62]:
```
print(new_value[2])
```

```
('n', 9)
```

[63]:
```
# Update a value

things.update({'key3':1999})
things
```

[63]:
```
{'key1': 'thing1', 'key2': 'thing2', 'key3': 1999, 'key4': 'thing4'}
```

[64]:
```
things.items()
```

[64]:
```
dict_items([('key1', 'thing1'), ('key2', 'thing2'), ('key3', 1999), ('key4',
'thing4')])
```

[65]:
```
# Remove a key:value pair

things = {'key1':'thing1','key2':'thing2','key3':1999, 'key4':'thing4'}

#things.pop('key3')
#things

# things.popitem()   # popitem will only remove the last item

things.clear()     # empties the dictionary
things
```

[65]:
```
{}
```

[66]:
```
# A preview of loops
things = {'key1':'thing1','key2':'thing2','key3':1999, 'key4':'thing4'}

for xyz in things:
    print(xyz)          # print the keys
#    print(things[xyz])  # print the values

#for xyz in things.keys():
#    print(xyz)

#for xyz in things.values():
#    print(xyz)
```

```
key1
key2
key3
key4
```

[67]:
```
for a,b in things.items():
    print(a,b)
```

```
key1 thing1
```

```
key2 thing2
key3 1999
key4 thing4
```

2.6 Dictionary Exercise - 15 minutes

https://www.w3schools.com/python/python_dictionaries_methods.asp

1. Create a dictionary with 5 key:value pairs
2. Get the value of the thrid pair
3. Update the value of the third pair
4. Use items to return a tuple of the key:value pairs
5. Return a list of all the values
6. Use the setdefault method. Used when a key is not in the dictionary.

[68]:
```
# Dictionary Exercise Here...
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

2.7 Arrays & Dataframes

Arrays and Dataframes will be discussed after numpy and pandas are introduced.

3 Control Flows

3.1 If/then/else Statements

[69]:
```
# The basic format... If (test is true): do something
```

[70]:
```
a = 100
b = 95
x = a > b
if x == True: print('yes')
```

```
yes
```

[71]:
```
things = {'key1':'thing1','key2':'thing2','key3':1999, 'key4':'thing4'}
if things.get('key1') == 'thing1': print('it worked')
```

```
# What happens if it fails?
#if things.get('key1') == 'thing2': print('it worked')
```

```
it worked
```

[72]:
```
things = {'key1':'thing1','key2':'thing2','key3':1999, 'key4':'thing4'}
if things.get('key1') == 'thing2':
    print('it worked')
else:                                          # Don't forget the colon!!!!
    print('key1 does not hold thing1')
```

```
key1 does not hold thing1
```

[73]:
```
things = {'key1':'thing1','key2':'thing2','key3':1999, 'key4':'thing4'}
if things.get('key1') == 'thing2':
    print('key1 holds thing2')
elif things.get('key1') == 'thing3':
    print('key1 holds thing3')
elif things.get('key1') == 'thing1':   # after running this, change thing1 to␣
 ↪thing7 to get else to work
    print('key1 holds thing1')
else:
    print('key1 must hold thing4')
```

```
key1 holds thing1
```

3.2 Loops

There are two loop commands: for & while.

3.2.1 for loops

[74]:
```
letters = ['a', 'b', 'c', 'd']
letters
```

[74]:
```
['a', 'b', 'c', 'd']
```

[75]:
```
# Basic format for 'any name you want' in 'the collection you are using': do␣
 ↪something

letters = ['a', 'b', 'c', 'd']
for item in letters: print(item)
```

```
a
b
```

```
c
d
```

[76]:
```
letters = ['a', 'b', 'c', 'd']
for item in letters:
    print(item+'xyz')
```

```
axyz
bxyz
cxyz
dxyz
```

[77]:
```
letters = ['a', 'b', 'c', 'd']
for item in letters:
    x = (item+'xyz')
```

[78]:
```
# Loop through a string

for each in 'Su-Sussudio':
    print(each)
```

```
S
u
-
S
u
s
s
u
d
i
o
```

[79]:
```
# Using a tuple

letters_t = ('a', 'b', 'c', 'd')
for item in letters:
   print(item)
```

```
a
b
c
d
```

[80]:
```
# Using a set

letters_t = {'a', 'b', 'c', 'd'}
for item in letters:
```

```
    print(item)
```

```
a
b
c
d
```

[81]:
```
months = [1,2,3,4,5,6,7,8,9,10,11,12]
months
```

[81]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

[82]:
```
for item in months:
    if item > 10:
        print(item)
```

```
11
12
```

[83]:
```
new_set = {}
for item in months:
    if item > 7:
        new_set[item] = item+100
        print(new_set)
```

```
{8: 108}
{8: 108, 9: 109}
{8: 108, 9: 109, 10: 110}
{8: 108, 9: 109, 10: 110, 11: 111}
{8: 108, 9: 109, 10: 110, 11: 111, 12: 112}
```

[84]:
```
type(new_set)
```

[84]:
```
dict
```

3.2.2 while loops

Continue to execute as long as a condition is True.

[85]:
```
i = 1
while i < 6:
    print(i)
    i += 1  # i = i + 1
```

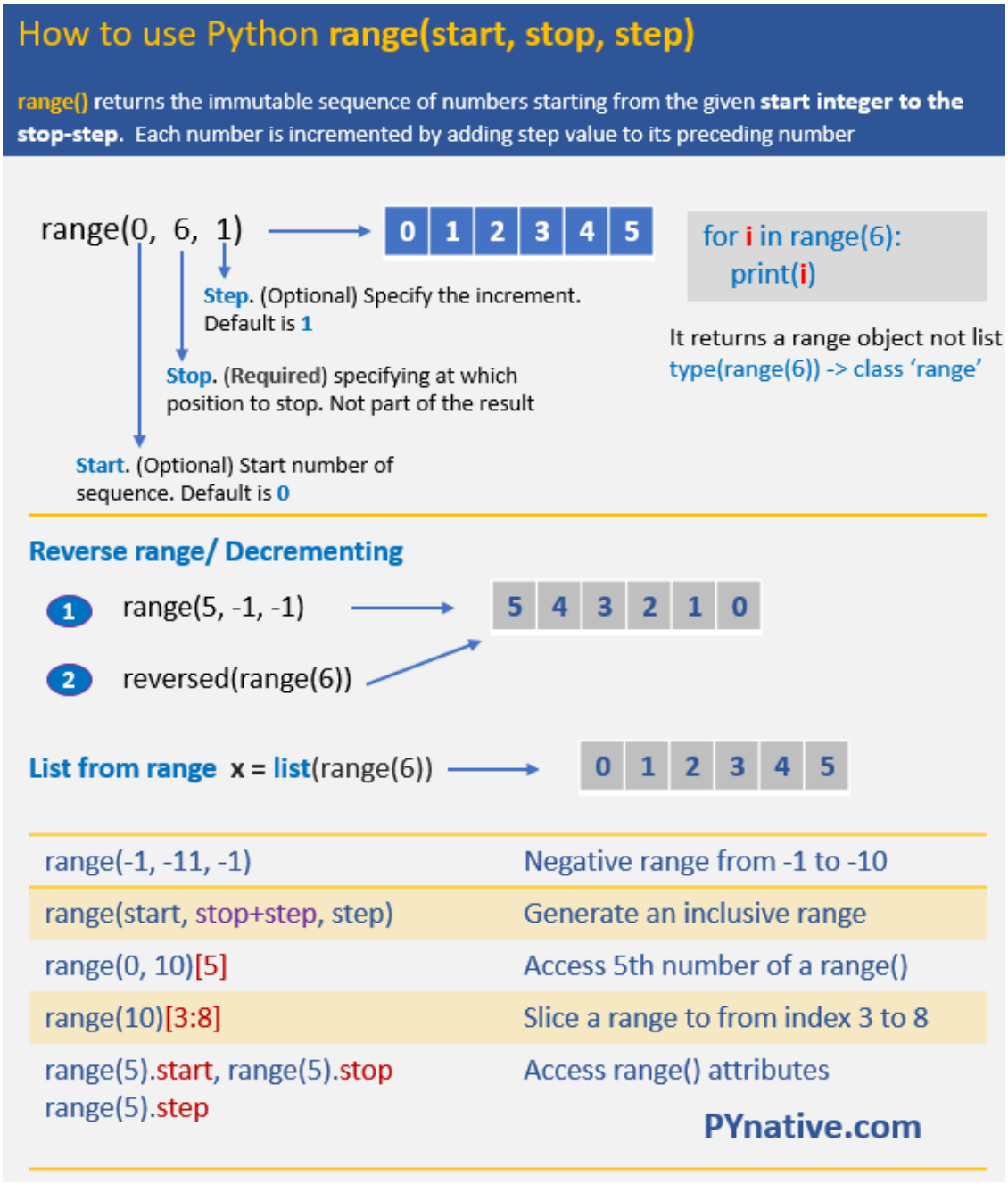
```
1
2
3
```

```
4
5
```

[86]:
```
i = 1
while i < 6:
    print(i)
    i += 1
else:
    print('we are done')
```

```
1
2
3
4
5
we are done
```

3.2.3 range()



https://pynative.com/python-range-function/

[87]:
```
for each in range(5):
    print(each)
```

```
0
1
2
3
4
```

[88]:
```
x = list(range(5,-1,-1))
x
```

[88]:
```
[5, 4, 3, 2, 1, 0]
```

[89]:
```
cities = ['Boston','Chicago','Atlanta','NYC','LA']
for each in range(len(cities)):  # Determining how many items are in the list
    print(each, cities[each])
```

```
0 Boston
1 Chicago
2 Atlanta
3 NYC
4 LA
```

[90]:
```
for each in cities:
    print(each)
```

```
Boston
Chicago
Atlanta
NYC
LA
```

[91]:
```
for each in range(2):  # Determining how many items are in the list
    print(each, cities[each])
```

```
0 Boston
1 Chicago
```

3.2.4 list comprehension

List comprehension is the use of compact syntax to create a list from another list or a string. It is faster than using a loop.

[92]:
```
# Create a list of even numbers

even_nums = []
for x in range(21):
    if x%2 == 0:
        even_nums.append(x)
print(even_nums)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

[93]:
```
even_nums = [x for x in range(21) if x%2 == 0]
print(even_nums)
```

```
# In the above example, [x for x in range(21) if x%2 == 0] returns a new list␣
 ↪using the list comprehension.
# First, it executes the for loop for x in range(21) if x%2 == 0. The element x␣
 ↪would be returned if the
# specified condition if x%2 == 0 evaluates to True.
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

3.3 Functions

A function is a block of code. In order for the code to be executed it must be called.

The keyword def introduces a function definition. It must be followed by the function name and the parenthesized list of formal parameters. The statements that form the body of the function start at the next line, and must be indented.

[94]:
```
months = [1,2,3,4,5,6,7,8,9,10,11,12]

for item in months:
    if item > 10:
        print(item)
```

```
11
12
```

[95]:
```
def month_test(xyz):         # xyz is a parameter.
    for item in xyz:
        if item > 5:
            print(item)
```

[96]:
```
month_test(months)          # months is the arguement sent to the function.
```

```
6
7
8
9
10
11
12
```

3.3.1 Arguements (*args) and keyword arguements (kwargs)

[97]:
```
def city_list1(city):
    print('The name of the city is ' + city)

city_list1('NY')
```

```
The name of the city is NY
```

[98]:
```
# Maybe we don't know how long the list will be but we know we always want the␣
 ↪value in second position.

def city_list1(*city):
    print('The name of the city is ' + city[1])

city_list1('NY','LA', 'Chicago')
```

```
The name of the city is LA
```

[99]:
```
# Keyword arguements
def city_list1(city1, city3, city2):
    print('The name of the city is ' + city2)

city_list1(city3 = 'NY',city2 = 'LA', city1 = 'Chicago')
```

```
The name of the city is LA
```

3.4 Lambda functions

A lambda function is anonymous. That means it does not have a name.

[100]:
```
def times2(var):
    return var*2

times2(5)
```

[100]:
```
10
```

[101]:
```
# Use the keyword lambda

lambda var: var*2
```

[101]:
```
<function __main__.<lambda>(var)>
```

[102]:
```
x = lambda var: var*2
x(5)                    # pass in the variable
```

[102]:
```
10
```

[103]:
```
y = lambda a, b: a+b
y(10,20)
```

[103]:
```
30
```

Lambda functions are used along with built-in functions like filter(), map() etc.

map() is a built-in function that transform all the items in an iterable without using an explicit for loop.

[104]:
```
months = [1,2,3,4,5,6,7,8,9,10,11,12]

new_list = list(filter(lambda x: (x%2 == 0) , months))
new_list2 = list(map(lambda x: x*2 , months))

print(new_list)
print(new_list2)
```

```
[2, 4, 6, 8, 10, 12]
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24]
```

3.5 Exerecise - 10 minutes

- Create a list that contains the names of 10 different fruits.
- Create a function that converts each value in the list to upper case…. str.upper()
- Apply the function

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

[105]:
```
# This is not the solution

fruits=['grapes','apples','oranges','pineapples','kiwi']
new_fruits = list(map(lambda x: str.upper(x) , fruits))
print(new_fruits)
```

```
['GRAPES', 'APPLES', 'ORANGES', 'PINEAPPLES', 'KIWI']
```

4 Reading data files

[106]:
```
#f = open("data/diabetes.csv",'r')
#print(f.read())
#f.close()
```

[107]:
```
#f = open("data/diabetes.csv",'r')
#print(f.readline())
#print(f.readline())
#print(f.readline())
#f.close()
```