Python-Basics

January 18, 2021

0.0.1 Beginning

Following the tradition, we start with a test message.

[1]:
```
print("Hello, World!")
```

```
Hello, World!
```

Here `print()` is a built-in function. "Hello, World!" is a string. (Explained later)

0.0.2 Variables

A variable is a named location used to store data in the memory.

[5]:
```
a=2
b=0.3
g1="Hello"
g2="World!"
print(a, b, a+b, g1, g2, g1+g2)
```

```
2 0.3 2.3 Hello World! HelloWorld!
```

Every object in Python has a type.

[9]:
```
print(type(a), type(b), type(a+b), type(g1))
```

```
<class 'int'> <class 'float'> <class 'float'> <class 'str'>
```

0.0.3 Operators

Operators are used to perform operations on variables and values.

- Arithmetic operators: `+` (addition), `-` (subtraction), `*` (multiplication), `/` (division), `**` (exponent), `//` (floor division), `%` (remainder)
- Comparison operators: `==` (equal to), `!=` (not equal to), `<` (less than), `<=` (less than or equal to), `>` (greater than), `>=` (greater than or equal to)
  - They return either `True` or `False`.
- Assignment operators: `=` (assign a value), `+=` (add and assign), `-=` (subtract and assign), `*=` (multiply and assign), `/=` (division and assign)
  - similarly, `//=`,`%=`,`**=`

[11]:
```
a=150; b=27
print(a+b, a-b, a*b, a/b)
```

```
177 123 4050 5.555555555555555
```

[14]:
```
a=17; b=5
print(a**b, a//b, a%b)
```

```
1419857 3 2
```

[99]:
```
g="good "; d="day "
print(g+d, g*3, d*2)
```

```
good day  good good good  day day
```

[21]:
```
x=5
x+=5; print(x)
x-=3; print(x)
x*=6; print(x)
x/=3; print(x)
```

```
10
7
42
14.0
```

[24]:
```
y=10
y//=3; print(y)
y**=2; print(y)
y%=4; print(y)
```

```
3
9
1
```

[37]:
```
g='good '
g+='day '; print(g)
g*=3; print(g)
```

```
good day
good day good day good day
```

[44]:
```
print(5==7, 4<7, 4<5<6, 3+2==5, 'good'=='bad')
```

```
False True True True False
```

0.0.4 Data types

- List(**`list`**): [item1, item2, ...]

- Tuple(**`tuple`**): (item1, item2, …)
- Dictionary(**`dict`**): { key1:value1, key2:value2, … }

[2]:
```
li = [1, 3, 5, 7, 9, 'dog', 'cat'] # list
tu = (2, 4, 6, 8, 10) # tupule
di = {'name':'Alice', 'age':21,\
      'favorite_fruit':['apple','banana','cherry']} # dictionary
# \ is used to seprate lines.
```

[3]:
```
li*2
```

[3]:
```
[1, 3, 5, 7, 9, 'dog', 'cat', 1, 3, 5, 7, 9, 'dog', 'cat']
```

[4]:
```
li += [11, 'hamster']; print(li)
```

```
[1, 3, 5, 7, 9, 'dog', 'cat', 11, 'hamster']
```

[5]:
```
tu *=3; print(tu)
```

```
(2, 4, 6, 8, 10, 2, 4, 6, 8, 10, 2, 4, 6, 8, 10)
```

[7]:
```
di.update({'name':'Betty'}); print(di)
```

```
{'name': 'Betty', 'age': 21, 'favorite_fruit': ['apple', 'banana', 'cherry']}
```

Here `.update()` is a method. (Explained later)

0.0.5 Accessing data

- Lists and tuples are indexed by integers, *starting at 0* for the first item.
  - To access a range of items, one can use **slicing**.
    - `list[start:stop]` items start through stop-1
    - `list[start:]` items start through the rest of the list
    - `list[:stop]` items from the beginning through stop-1
    - `list[:]` a copy of the whole list
  - Negative indexing starts at the back of a sequence.
- A string can be accessed in the same way.
- Dictionaries are indexed by their keys.

[23]:
```
li_odd=[1,3,5,7,9,11,13,15]
```

[54]:
```
li_odd[0], li_odd[3], li_odd[-1]
```

[54]:
```
(1, 7, 15)
```

[28]:
```
print(li_odd[:])
print(li_odd[2:5])
print(li_odd[3:])
print(li_odd[:2])
```

```
print(li_odd[-2:])
print(li_odd[:-1])
```

```
[1, 3, 5, 7, 9, 11, 13, 15]
[5, 7, 9]
[7, 9, 11, 13, 15]
[1, 3]
[13, 15]
[1, 3, 5, 7, 9, 11, 13]
```

[53]:
```
hi='How are you?'
hi[0],hi[4],hi[-1], hi[1:-1]
```

[53]:
```
('H', 'a', '?', 'ow are you')
```

[8]:
```
di = {'name':'Alice', 'age':21,\
      'favorite_fruit':['apple','banana','cherry']}
```

[9]:
```
di['name']
```

[9]:
```
'Alice'
```

[11]:
```
di['favorite_fruit'][2]
```

[11]:
```
'cherry'
```

0.0.6 Built-in functions

A **function** performs an action and/or return a value.

Examples of built-in functions: - **`print()`**, **`type()`** - functions for lists or tuples: **`len()`** (length), **`sum()`**, **`min()`**, **`max()`** - functions for numbers: **`abs()`** (absolute value), **`round()`** - type conversion: **`int()`**, **`str()`**, **`float()`**, **`list()`** - **`input()`** (taking an input string from the user)

You can define your own functions. (Explained later)

[45]:
```
li_odd=[1,3,5,7,9,11,13,15]
```

[52]:
```
len(li_odd), sum(li_odd), max(li_odd), abs(-7)
```

[52]:
```
(8, 64, 15, 7)
```

[23]:
```
a=3.14159; b=-a
round(a), b, abs(b), round(b)
```

[23]:
```
(3, -3.14159, 3.14159, -3)
```

[24]:
```
a=3; b=float(a); c=str(a)
type(a), type(b), type (c)
```

[24]:
```
(int, float, str)
```

[27]:
```
name=input("Enter your name: ")
print("Hi, "+name+"!")
```

```
Enter your name: Tom
Hi, Tom!
```

0.0.7 Methods

An object has its **methods** which are functions available for the object. These are accessed by the format `object.method()`.

0.0.8 Some methods on string objects

`.capitalize()`, **`.upper()`** (upper case), **`.lower()`** (lower case), **`.count(substring)`**, **`.replace(old, new)`**

[46]:
```
sentence = 'aMyloidoSis Is a disEase.'
```

[47]:
```
sentence.capitalize(), sentence.upper(), sentence.lower()
```

[47]:
```
('Amyloidosis is a disease.',
 'AMYLOIDOSIS IS A DISEASE.',
 'amyloidosis is a disease.')
```

[48]:
```
sentence.count('s'), sentence.count('is')
```

[48]:
```
(4, 2)
```

[51]:
```
sentence.replace('is', '!$')
```

[51]:
```
'aMyloidoS!$ Is a d!$Ease.'
```

0.0.9 Some methods on list objects

`.append(item)`, **`.extend([item1, item2, ...])`**, **`.remove(item)`**

[66]:
```
li=[1,3,5,7,9]
li.append(17)
li
```

[66]:
```
[1, 3, 5, 7, 9, 17]
```

[67]:
```
li.extend([19,23])
li
```

[67]:
```
[1, 3, 5, 7, 9, 17, 19, 23]
```

[68]:
```
li.remove(5)
li
```

[68]:
```
[1, 3, 7, 9, 17, 19, 23]
```

0.0.10 Some methods on dict objects

`.update(dict1)` (to update with dict1), **`.keys()`** (list of keys), **`.values()`** (list of values)

[12]:
```
di = {'name':'Alice', 'age':21,\
      'favorite_fruit':['apple','banana','cherry']}
```

[14]:
```
di.update({'name':'Betty', 'major': 'math' })
```

[15]:
```
di
```

[15]:
```
{'name': 'Betty',
 'age': 21,
 'favorite_fruit': ['apple', 'banana', 'cherry'],
 'major': 'math'}
```

[16]:
```
di.keys()
```

[16]:
```
dict_keys(['name', 'age', 'favorite_fruit', 'major'])
```

[17]:
```
di.values()
```

[17]:
```
dict_values(['Betty', 21, ['apple', 'banana', 'cherry'], 'math'])
```

0.0.11 `if` statements and `while` loops

An **`if` statement** tests a condition and performs some actions if the condition evaluates to `True`. If the condition evaluates to `False`, alternative actions can be taken by **`elif`** and/or **`else`**.

Warning: Indentation is extremely important!

A **`while` loop** will keep performing some actions as long as its condition evaluates to `True`.

[115]:
```
num = int(input("Enter an integer greater than 10: "))

if num > 10:
    print("Great job!")
```

```
Enter an integer greater than 10: 7
```

Here `int()` and `input()` are built-in functions.

[113]:
```
num = int(input("Enter an integer: "))

if num%2 == 0:
    print("Even number")
else:
    print("Odd number")
```

```
Enter an integer: -17
Odd number
```

[112]:
```
num = int(input("Enter an integer: "))

if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

```
Enter an integer: -23
Negative number
```

[120]:
```
num = int(input("Enter a positive integer: "))

sum = 0
i = 1

while i <= num:
    sum = sum + i
    i = i+1

print("The sum from 1 to %d is " % num + str(sum) + ".")
```

```
Enter a positive integer: 99
The sum from 1 to 99 is 4950.
```

%d is used to refer to a variable of type **int** which follows after %. Similarly, one can use **%s** and **%f**.

0.0.12 `for` loops

Actions can be iterated over a collection of items using a **`for` loop**. The strings, lists, tuples and dictionaries are all **iterable** containers. It is also common to use **`range()`**. - A `for` loop will go through the specified container, one item at a time. - **`break`** can be used to terminate a loop.

[121]:
```
li_odd = [1,3,5,7,9,11,13,15]
```

```
sum = 0

for n in li_odd:
    sum += n

print("The sum is "+str(sum)+".")
```

```
The sum is 64.
```

[129]:
```
print(list(range(10)))
print(list(range(3,10)))
print(list(range(1,10,2)))
print(list(range(10,2)))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[3, 4, 5, 6, 7, 8, 9]
[1, 3, 5, 7, 9]
[]
```

Here **list( )** converts the type from **range** to **list**.

[130]:
```
num = int(input("Enter a positive integer: "))
sum =0
for i in range(1,num+1):
    sum += i

print("The sum from 1 to %d is " % num + str(sum) + ".")
```

```
Enter a positive integer: 21
The sum from 1 to 21 is 231.
```

[56]:
```
for char in "How are you?":
    print(char)
```

```
H
o
w

a
r
e

y
o
u
?
```

[61]:
```
for char in "How are you?":
    if char == "y":
        break
    print(char, end="") # This prints on the same line.
```

```
How are
```

0.0.13 List comprehensions

There is a simple way to generate a list with a single line of code using **list comprehensions**.

[140]:
```
[a**2 for a in range (1,10)]
```

[140]:
```
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

[145]:
```
[a**2 for a in range (1,20) if a %3 ==0]
```

[145]:
```
[9, 36, 81, 144, 225, 324]
```

[146]:
```
[a**2 if a %3 ==0 else a for a in range (1,20)]
```

[146]:
```
[1, 2, 9, 4, 5, 36, 7, 8, 81, 10, 11, 144, 13, 14, 225, 16, 17, 324, 19]
```

0.0.14 Copying a list

This brings about a common mistake.

[150]:
```
li_a=[1,2,3]
li_b=li_a  # The location of memory is copied.
li_b.append(4)
li_a, li_b
```

[150]:
```
([1, 2, 3, 4], [1, 2, 3, 4])
```

[151]:
```
li_a=[1,2,3]
li_b=li_a[:]  # Only the values are copied.
li_b.append(4)
li_a, li_b
```

[151]:
```
([1, 2, 3], [1, 2, 3, 4])
```

[57]:
```
li_a=[1,2,3]
li_b=li_a.copy()  # Only the values are copied.
li_b.append(4)
li_a, li_b
```

[57]:
```
([1, 2, 3], [1, 2, 3, 4])
```

0.0.15 How to define a function

A **function** is a group of statements that perform a specific task. You use **`def`** to create a function. A result can be returned from a function using a **`return`** statement.

[62]:
```
def print_name(name):
    print("Hi! I am "+name+".")

print_name('John')
print_name('Cynthia')
```

```
Hi! I am John.
Hi! I am Cynthia.
```

[2]:
```
def divisors(num):
    dv=[]
    for i in range(1,num+1):
        if num%i ==0:
            dv.append(i)
    return dv

divisors(6), divisors(23), divisors(1234), divisors(2021)
```

[2]:
```
([1, 2, 3, 6], [1, 23], [1, 2, 617, 1234], [1, 43, 47, 2021])
```

[5]:
```
def is_prime(num):
    dv=divisors(num)
    if len(dv) == 2:
        return True
    else:
        return False

is_prime(5), is_prime(12), is_prime(2311)
```

[5]:
```
(True, False, True)
```

0.0.16 Recursive functions

A function that calls itself is a **recursive function**. Every recursive function must have a condition that stops the recursion in order to avoid an infinite loop.

[3]:
```
def factorial(num):
    if num == 1:
        return 1
    else:
        return num*factorial(num-1)

factorial(5), factorial(12)
```

[3]:
```
(120, 479001600)
```

[7]:
```
def fibonacci(num):
    if num == 0:
        return 0
    elif num == 1:
        return 1
    else:
        return fibonacci(num-1)+fibonacci(num-2)

[fibonacci(i) for i in range(15)]
```

[7]:
```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
```

0.0.17 Modules

A module is a file containing Python statements and definitions. One can **import** a module.

[8]:
```
pi
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-8-f84ab820532c> in <module>
----> 1 pi

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

[12]:
```
import math

math.pi, math.e
```

[12]:
```
(3.141592653589793, 2.718281828459045)
```

[13]:
```
import math as m

m.pi, m.e
```

[13]:
```
(3.141592653589793, 2.718281828459045)
```

One can import specific names from a module. In such a case, a dot after the moudle name is not needed.

[15]:
```
from math import pi,e

pi, e
```

[15]:
```
(3.141592653589793, 2.718281828459045)
```

We can import all names from a module.

[59]:
```
from math import *
```

[60]:
```
gamma(1/2) == sqrt(pi)
```

[60]:
```
True
```

It is true that $\Gamma\left(\frac{1}{2}\right) = \sqrt{\pi}$ where $\Gamma(x) = \int_0^\infty t^{x-1}e^t dt$ is the *gamma* function.