Data Management: Introduction to Python

Michel Coppée

March

Overview

Introduction

What is Python ?

Why Python ?

How to use Python ?

Let's guess

Code

```
wealth = 0
income = 150
invoice = 100

# Compute wealth
net_income = income - invoice

if net_income > 0:
    print("You're richer !")

wealth = wealth + net_income
print(wealth)
```

Result

```
You're richer !
50
```

Data type - Numbers

Code

```
net_income = 50
type(net_income)
type(19.95)
```

Result

int
float

Arithmetic operators

Operator	Name	Description
a + b	Addition	Sum of a and b
a - b	Subtraction	Difference of a and b
a * b	Multiplication	Product of a and b
a / b	True division	Quotient of a and b
a // b	Floor division	Quotient of a and b, removing fractional parts
a % b	Modulus	Integer remainder after division of a by b
a ** b	Exponention	a raised to the power of b
-a	Negation	The negative of a

Build-in functions for numbers

Code

```
print(min(1,2,3))
print(max(1,2,3))
print(abs(-32))

print(float(10))
print(int(3.33))

print(int('807') + 1)
```

Result

```
1
3
32
10.0
3
808
```

Getting help

Code

```
help(round)
```

Result

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

round(number, ndigits=None)
    Round a number to a given precision in decimal digits.

    The return value is an integer if ndigits is omitted or None.  Otherwise
    the return value has the same type as the number.  ndigits may be negative.
```

Defining functions

Code

```
def compute_wealth(income, invoice):
    wealth = income - invoice
    return wealth

print(
    compute_wealth(200, 100),
    compute_wealth(50, 100),
    compute_wealth(300, 225),
)
```

Result

```
100
-50
75
```

Functions that don't return

Code

```
wealth = 0

def compute_wealth(income, invoice):
    wealth = income - invoice

print(compute_wealth(100, 50))
```

Result

None

Code

```
compute_wealth(100,50)
print(wealth)
```

Result

50

Default arguments

Code

```
help(print)
```

Result

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

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Default arguments

Code

```
print(1, 2, 3)
```

Result

1 2 3

Code

```
print(1, 2, 3, sep=' < ')
```

Result

1 < 2 < 3

Default arguments

Code

```
def greet(who="Students"):
    print("Hello,", who)

greet()
greet(who="Python")
# (In this case, we don't need to specify the name of the argument,
# because it's unambiguous.)
greet("World")
```

Result

Hello, Students
Hello, Python
Hello, World

Booleans

Code

```
x = True
print(x)
print(type(x))
```

Result

```
True
<class 'bool'>
```

Boolean operators

a == b	a equal to b		a != b	a not equal to b
a < b	a less than b		a > b	a greater than b
a <= b	a less than or equal to b		a >= b	a greater than or equal to b

Code

```
def can_run_for_president(age):
    # The US Constitution says you must be at least 35 years old
    return age >= 35

print("Can a 19-year-old run for president?", can_run_for_president(19))
print("Can a 45-year-old run for president?", can_run_for_president(45))
```

Result

Can a 19-year-old run for president? False
Can a 45-year-old run for president? True

Boolean operators

Code

```
3.0 == 3
'3' == 3
```

Result

True
False

Combining Boolean Values

Keywords

and, or, not

Code

```
def can_run_for_president(age, is_natural_born_citizen):
    # The US Constitution says you must be a natural born citizen *and*
    # at least 35 years old
    return is_natural_born_citizen and (age >= 35)

print(can_run_for_president(19, True))
print(can_run_for_president(55, False))
print(can_run_for_president(55, True))
```

Result

False
False
True

Conditionals

Code

```
def inspect(x):
    if x == 0:
        print(x, "is zero")
    elif x > 0:
        print(x, "is positive")
    elif x < 0:
        print(x, "is negative")
    else:
        print(x, "is unlike anything I've ever seen...")

inspect(0)
inspect(-15)
```

Result

0 is zero
-15 is negative

Boolean conversion

Code

```
print(bool(1))
print(bool(0))
print(bool("asf"))
print(bool(""))
```

Result

```
True
False
True
False
```

Lists

Code

```
primes = [2, 3, 5, 7]

planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn',
           'Uranus', 'Neptune']

hands = [
    ['J', 'Q', 'K'],
    ['2', '2', '2'],
    ['6', 'A', 'K'], # (Comma after the last element is optional)
]

hands = [['J', 'Q', 'K'], ['2', '2', '2'], ['6', 'A', 'K']]

mixed_list = [32, 'raindrops on roses', help]
```

Lists - Indexing

Code

```
print(planets[0])
print(planets[1])
print(planets[-1])
print(planets[-2])
```

Result

'Mercury'
'Venus'
'Neptune'
'Uranus'

Lists - Slicing

Code

```
print(planets[0:3])
print(planets[:3])
print(planets[3:])
print(planets[1:-1])
print(planets[-3:])
```

Result

['Mercury', 'Venus', 'Earth']
['Mercury', 'Venus', 'Earth']
['Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
['Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus']
['Saturn', 'Uranus', 'Neptune']

Lists - Changing lists

Code

```
planets[3] = 'Malacandra'
print(planets)
```

Result

['Mercury', 'Venus', 'Earth', 'Malacandra', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']

Lists - functions

Code

```
print(len(planets))
print(sorted(planets))

primes = [2, 3, 5, 7]
print(sum(primes))
print(max(primes))
```

Result

```
8
['Earth', 'Jupiter', 'Mars', 'Mercury', 'Neptune', 'Saturn', 'Uranus', 'Venus']
17
7
```

List - Methods

Everything in Python is an Object. It means that it carries attributes and methods.

Code

```
planets.append('Pluto')
print(planets)

planets.pop()
print(planets)
```

Result

['Earth', 'Jupiter', 'Mars', 'Mercury', 'Neptune', 'Saturn', 'Uranus', 'Venus', 'Pluto']
['Earth', 'Jupiter', 'Mars', 'Mercury', 'Neptune', 'Saturn', 'Uranus', 'Venus']

List - Searching

Code

```
print(planets.index('Earth'))
print(planets.index('Pluto'))
print('Earth' in planets)
print('Calbefraques' in planets)
```

Result

'2'

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipykernel_20/2263615293.py in <module>
----> 1 planets.index('Pluto')

ValueError: 'Pluto' is not in list
```

True
False

More on attributes and methods

Code

```
help(planets)
```

Result

```
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:
```

```
|  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.
```

Tuples

Code

```
t = (1, 2, 3)
t = 1, 2, 3 # equivalent to above
print(t)

t[0] = 100
```

Result

(1, 2, 3)

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_20/816329950.py in <module>
----> 1 t[0] = 100

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

Tuples

Code

```
x = 0.125
print(x.as_integer_ratio())
```

Result

(1, 8)

Code

```
numerator, denominator = x.as_integer_ratio()
print(numerator / denominator)
```

Result

0.125

Loops

Code

```
planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn',
           'Uranus', 'Neptune']
for planet in planets:
    print(planet, end=' ')
```

Result

Mercury Venus Earth Mars Jupiter Saturn Uranus Neptune

Code

```
multiplicands = (2, 2, 2, 3, 3, 5)
product = 1
for mult in multiplicands:
    product = product * mult
print(product)
```

Result

360

Loops

Code

```
s = 'Here iS oNe sEnTenCe'
msg = ''
# print all the uppercase letters in s, one at a time
for char in s:
    if char.isupper():
        print(char, end='')
```

Result

HSNETC

Loops - range()

Code

```
for i in range(5):
    print("Doing important work. i =", i)
```

Result

Doing important work. i = 0
Doing important work. i = 1
Doing important work. i = 2
Doing important work. i = 3
Doing important work. i = 4

Loops - while

Code

```
i = 0
while i < 10:
    print(i, end=' ')
    i += 1 # increase the value of i by 1
```

Result

```
0 1 2 3 4 5 6 7 8 9
```

List comprehensions

Code

```
squares = []
for n in range(10):
    squares.append(n**2)
print(squares)
```

Code

```
squares = [n**2 for n in range(10)]
print(squares)
```

Result

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

We can also add an "if" condition.

Strings

Code

```
print('Pluto's a planet!')
print("Pluto's a planet!")
```

Result

```
  File "/tmp/ipykernel_19/1561186517.py", line 1
    'Pluto's a planet!'
           ^
SyntaxError: invalid syntax
```

Pluto's a planet!

Strings

What you type...	What you get	example	print(example)
\'	'	'What\'s up?'	What's up?
\"	"	"That's \"cool\""	That's "cool"
\\	\	"Look, a mountain: /\\"	Look, a mountain: /\
\n		"1\n2 3"	1 2 3

Strings are sequences

Code

```
planet = 'Pluto'
print(planet[0])

print(planet[-3:])
print(len(planet))
print([char+'! ' for char in planet])

# strings are immutable. The following won't work:
planet[0] = 'B'
# planet.append doesn't work either
```

Result

```
'P'
'uto'
5
['P! ', 'l! ', 'u! ', 't! ', 'o! ']
```

Strings - Methods

Code

```
claim = "Pluto is a planet!"
print(claim.upper())
print(claim.lower())
print(claim.index('plan'))
print(claim.startswith(planet))
print(claim.endswith('dwarf planet'))
```

Result

'PLUTO IS A PLANET!'
'pluto is a planet!'
11
True
False

Strings - Methods

Code

```
words = claim.split()
print(words)
datestr = '1956-01-31'
year, month, day = datestr.split('-')
print(year, month, day)
print('/'.join([month, day, year]))
```

Result

['Pluto', 'is', 'a', 'planet!']
'1956' '31' '01'
'01/31/1956'

Strings - Methods

Code

```
planet = 'Pluto'
print(planet + ', we miss you.')
position = 9
print(planet + ", you'll always be the " + position + "th planet to me.")
print(planet + ", you'll always be the " + str(position) +
      "th planet to me.")
print("{}, you'll always be the {}th planet to me.".format(planet,
      position))
```

Result

'Pluto, we miss you.'

```
TypeError: can only concatenate str (not "int") to str
```

"Pluto, you'll always be the 9th planet to me."
"Pluto, you'll always be the 9th planet to me."

Dictionaries

Code

```
numbers = {'one':1, 'two':2, 'three':3}
print(numbers['one'])
numbers['eleven'] = 11
print(numbers)
numbers['one'] = 'Pluto'
print(numbers)
```

Result

```
1
'one': 1, 'two': 2, 'three': 3, 'eleven': 11
'one': 'Pluto', 'two': 2, 'three': 3, 'eleven': 11
```

Dictionaries

Code

```
planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn',
           'Uranus', 'Neptune']
planet_to_initial = {planet: planet[0] for planet in planets}
print(planet_to_initial)

print('Saturn' in planet_to_initial)
print('Betelgeuse' in planet_to_initial)
```

Result

{'Mercury': 'M', 'Venus': 'V', 'Earth': 'E', 'Mars': 'M', 'Jupiter': 'J', 'Saturn':'S',
'Uranus': 'U', 'Neptune': 'N'}
True
False

Dictionaries

Code

```
for k in numbers:
    print("{} = {}".format(k, numbers[k]))

# Access keys
print(planet_to_initial.keys())
#Access values
print(planet_to_initial.values())

#Access keys and values
for planet, initial in planet_to_initial.items():
    print("{} begins with \"{}\"".format(planet.rjust(10), initial))
```

Result

one = Pluto
two = 2
three = 3
eleven = 11

Working with external libraries

Code

```
import math
print(dir(math))
```

Result

```
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

Math module

Code

```
print(math.pi)
print(math.log(32,2))
```

Result

```
3.141592653589793
5.0
```

Remember that you call help() on any function or even on modules !

Modules - Warnings

Some modules can have variables referring to other modules.

Some modules define their own data types (others than ints, floats, bools, lists, strings, and dicts).

You can use the builtin functions:

1. type() - tells the type of a something
2. dir() - tells what you can do with something
3. help() - tells more about something / how to use it

Numpy module

Code

```
import numpy as np
print(dir(np))

# Examples
np.mean
np.median
np.min
np.max
np.random
np.sin
np.cos
```