Introduction to Data Science – Week 4

NETA LIVNEH

COURSE 55793

2019-2020

Last Week Recap

For loops

Functions

This week

Data types:

- Sets
- Dictionaries

Looping through containers

Strings Format

Data Types: Dictionary

A dictionary stores key-value pairs of objects.

each possible key is imputable and appears at most once in the collection (string, tuple).

Any key of the dictionary is associated (or mapped) to a value.

The values of a dictionary can be any Python data type.

Dictionaries are implemented as hash tables.

```
In [120]: grades = {'math': 90, 'statistics': 93, 'art': 85, 'sport': 70}
In [121]: grades
Out[121]: {'art': 85, 'math': 90, 'sport': 70, 'statistics': 93}

In [122]: grades['math']
Out[122]: 90
In [123]: grades['history'] = 87

In [124]: en_de = dict(red="rot", green="grün", blue ="blau", yellow="gelb")
```


Operations on Dictionary

Method	Description
dict.get(key, default)	For key key, returns value or default if key not in dictionary
dict.setdefault(key, default)	Similar to get(), but will set dict[key] = default if key is not already in dict
dict.fromkeys(seq, value)	Create a new dictionary with keys from seq and values set to value.
dict.keys()	Returns a view of dictionary dict's keys
dict.items()	Returns a view of dict's (key, value) tuple pairs
dict.values()	Returns a view of dictionary dict's values
dict.update(dict2)	Adds dictionary dict2's key-values pairs to dict
dict.pop(key, [default])	If *key* is in the dictionary, remove it and return its value, else return *default*. If *default* is not given and *key* is not in the dictionary, a KeyError is raised.

Looping through Dictionaries

```
In [128]: for en, de in en_de.items():
     ...:     print(en, 'is', de)
     ...:
red is rot
green is grün
blue is blau
yellow is gelb

In [129]: average = 0
In [130]: for value in grades.values():
     ...:     average += value
     ...:
In [131]: print(average/len(grades))
85.0
```

```
In [132]: total_length = 0
In [133]: for key in grades.keys():
     ...:     total_length += len(key)
     ...:

In [134]: print(total_length)
29
```


Data Types: Sets

A set is similar to a list but it cannot have multiple occurrences of the same element.

A set contains an *unordered* collection of *unique* and *immutable* objects (can't have lists).

Example:

```
In [115]: {1, 2, 3, 3, 4}
Out[115]: {1, 2, 3, 4}
In [116]: type(_)
Out[116]: set


In [117]: cities = set(("Paris", "Lyon", "London","Berlin","Paris","Birmingham"))
In [118]: cities
Out[118]: {'Berlin', 'Birmingham', 'London', 'Lyon', 'Paris'}
In [119]: empty_set = set()
```


Operations on Sets

Operation	Equivalent	Result
s.add(x)		Adds an element x, which has to be immutable, to a set s if it is not already there.
s.pop()		removes and returns an arbitrary set element. The method raises a KeyError if the set is empty.
s.discard(x)		Removed x from set s. If x is not in s there is no change.
s.remove(x)		Removed x from set s. The method raises a KeyError if x s not in set s.
s1. difference(s2)	s1-s2	Returns the difference of two or more sets as a new set
s1. difference_update(s2)	s1 = s1 - s2	Removes all elements of another set s2 from set s1.
s1.union(s2)	s1 | s2	Returns the union of two sets as a new set, i.e. all elements that are in either set s1 or s2.
s1.intersection(s2)	s1 & s2	Returns the intersection of set s1 and set s2 as a new set.
s1.isdisjoint(s2)		returns True if two sets have a null intersection
s1.issubset(s2)	s1 <= s2; s1 < s2	Returns True if s2 is a subset of s1. < for proper subset.
s1.superset(s2)	s1 >= s2; s1 > s2	Returns True if s2 is a superset of s1. < for proper superset.

Operations on Sets

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> fruit = set(basket)               # create a set without duplicates
>>> fruit
set(['orange', 'pear', 'apple', 'banana'])
>>> 'orange' in fruit                 # fast membership testing
True
>>> 'crabgrass' in fruit
False

>>> # Demonstrate set operations on unique letters from two words
...
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                                 # unique letters in a
set(['a', 'r', 'b', 'c', 'd'])
>>> a - b                             # letters in a but not in b
set(['r', 'd', 'b'])
>>> a | b                             # letters in either a or b
set(['a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'])
>>> a & b                             # letters in both a and b
set(['a', 'c'])
>>> a ^ b                             # letters in a or b but not both
set(['r', 'd', 'b', 'm', 'z', 'l'])
```


Sorted Function

The built-in function sorted(iterable, key, reverse) builds a new sorted list from an iterable.

It has two optional parameters: key, receives a function that determines the comparison key, and reverse, which sorts the list in a reversed order.

```
In [140]: sorted([5, 2, 3, 1, 4])
Out[140]: [1, 2, 3, 4, 5]

In [141]: sorted({1: 'D', 2: 'B', 3: 'B', 4: 'E', 5: 'A'})
Out[141]: [1, 2, 3, 4, 5]

In [142]: sorted("This is a test string from Andrew".split(), key=str.lower)
['a', 'Andrew', 'from', 'is', 'string', 'test', 'This']

In [143]: student_tuples = [('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10),]
In [144]: sorted(student_tuples, key=lambda student: student[2])   # sort by age
Out[144]: [('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```


Looping through Containers: Enumerate

The built-in function enumerate(seq) returns the position index and corresponding value at the same time.

Example:

```
In [146]: for i, v in enumerate(['tic', 'tac', 'toe']):
     ...:     print(i, v)
     ...:
0 tic
1 tac
2 toe
```


Looping through Containers: Zip

Make an iterator that aggregates elements from each of the iterables. Returns an iterator of tuples, where the i-th tuple contains the i-th element from each of the argument sequences or iterables. The iterator stops when the shortest input iterable is exhausted.

```
In [147]: x = [1,2,3]
In [148]: y = [4,5,6]
In [149]: zip(x,y)
Out[149]: <zip at 0x111811f88>

In [150]: list(_)
Out[150]: [(1, 4), (2, 5), (3, 6)]

In [151]: questions = ['name', 'quest', 'favorite color']
In [152]: answers = ['lancelot', 'the holy grail', 'blue']

In [153]: for q, a in zip(questions, answers):
     ...:     print('What is your {0}?  It is {1}.'.format(q, a))
     ...:
What is your name?  It is lancelot.
What is your quest?  It is the holy grail.
What is your favorite color?  It is blue.
```

Operations on Strings

Python text I/O is based on writing strings.

Any python value can be converted to string with the str() function.

There are three ways to format strings:

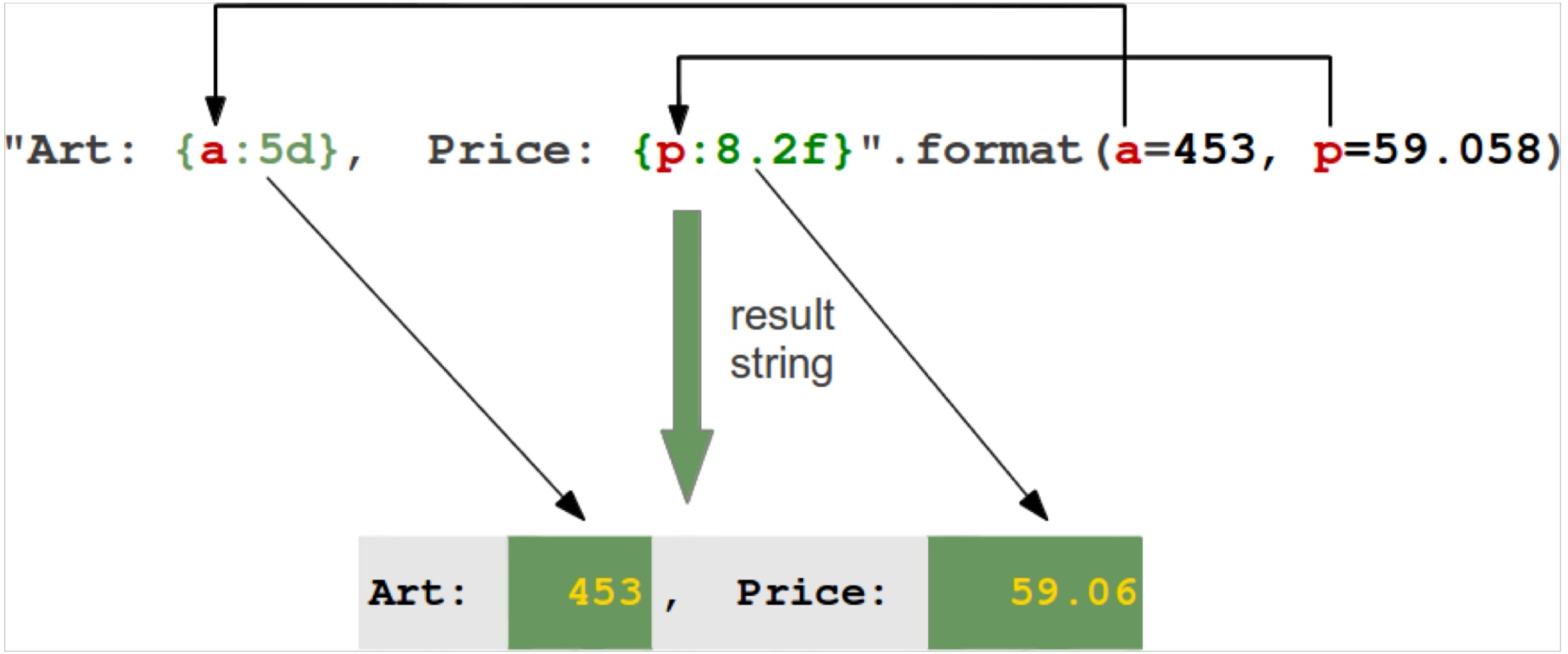
- slicing/concatenation:

```
In [173]: hi = "I'm Neta"
In [174]: ' '.join([hi[:3],"a string with", str(6), 'words'])
Out[174]: "I'm a string with 6 words"
```

- str.format() - using template

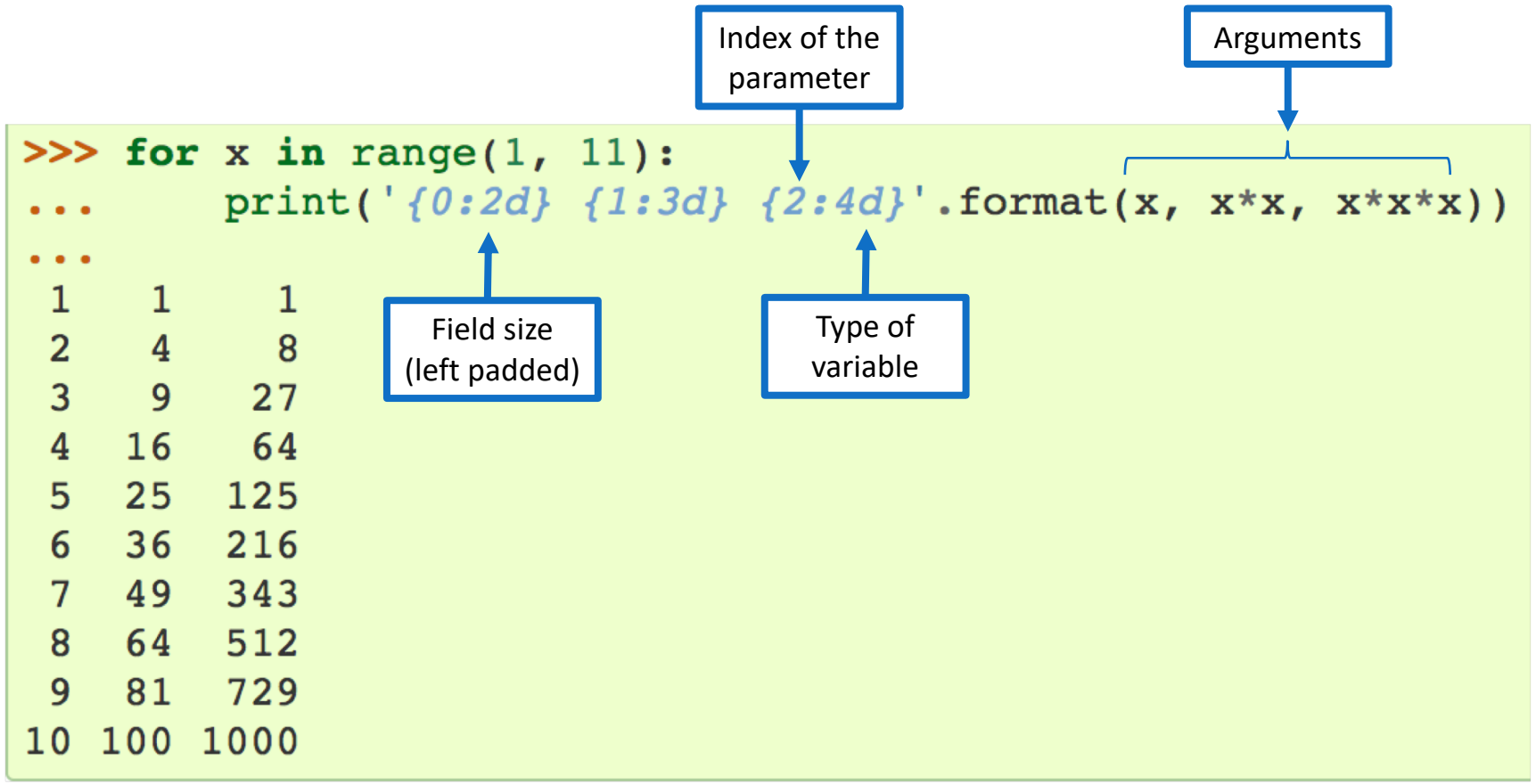
```
In [181]: 'The magic words are {0}{1}{0}'.format('abra', 'cad')
Out[181]: 'The magic words are abracadabra'
```


Operations on Strings – Format()



From: https://www.python-course.eu/python3_formatted_output.php

Operations on Strings – Format()



From: https://docs.python.org/3.8/library/string.html#string.Formatter

f-Stings – the new string format

```
>>> first = 'Monty'
>>> last = 'Python'
>>> f'Are you the {first} from {first} {last}?'
'Are you the Monty from Monty Python?’
>>> f'{134*27}'
'3618’
>>> f'Are you the {first} from {first.upper()} {last}?'
'Are you the Monty from MONTY Python?'
```