COP 3035

Intro Programming in Python

Summer 2024

Lecture 7 – part 1

Lab 4 - Due Date: 06/10/2024
Homework 2 - Due date: 06/07/2024
Homework 3 - Due date: 06/14/2024

Lecture 7 – part 2

Review

Name	**Type**	**Description**
Integers	int	Whole numbers, such as: **3** **300** **200**
Floating point	float	Numbers with a decimal point: 2.3 **4.6** **100.0**
Strings	str	Ordered sequence of characters: **"hello"** **'Sammy'** **"2000"** **"楽しい"**
Lists	list	Ordered sequence of objects: **[10,"hello",200.3]**
Dictionaries	dict	Unordered Key:Value pairs: **{"mykey" : "value" , "name" : "Frankie"}**
Tuples	tup	Ordered immutable sequence of objects: **(10,"hello",200.3)**
Sets	set	Unordered collection of unique objects: **{"a","b"}**
Booleans	bool	Logical value indicating **True** or **False**

Review

Dictionaries

```
d = {'key1':'value1','key2': 3,'key3': [12,23,33]}
n = d['key2']
d['key2'] = 5
d['New key'] = 'Hello'
d['key3'][1]
```

Dictionary methods

.keys(), values(), items()

Tuples:

(1,2)

Methods: .index(), .count()

Sets:

A = set([1,2,2,3])

{1,2,3}

Methods: set(), .add(), .remove(), .intersection(), .union(), .difference()

Boolean:

True, False

Operators: **and, or, not, all(), any(), in**

Method	Description	Example
`get()`	Returns the value for a specified key	`value = my_dict.get(key)`
`update()`	Updates the dictionary with elements from another dictionary or iterable	`my_dict.update(other_dict)`
`keys()`	Returns a view object of the dictionary's keys	`keys = my_dict.keys()`
`values()`	Returns a view object of the dictionary's values	`values = my_dict.values()`
`items()`	Returns a view object of the dictionary's key-value pairs	`items = my_dict.items()`
`pop()`	Removes the specified key and returns its value	`value = my_dict.pop(key)`
`popitem()`	Removes and returns the last inserted key-value pair	`key, value = my_dict.popitem()`
`setdefault()`	Returns the value of a key. If the key does not exist, inserts the key with a specified value	`value = my_dict.setdefault(key, default_value)`
`copy()`	Returns a shallow copy of the dictionary	`new_dict = my_dict.copy()`
`clear()`	Removes all items from the dictionary	`my_dict.clear()`

Lecture 7 – part 3

Control flow in python.
Conditional statements (if, elif, else).

If/else statement

- Syntax of the `if/else` statement

```
if True:
    # do something
    print(a)
else:
    # do something else
    print(b)
```

If/elif/else statement

- Syntax of the **`if/else`** statement

```
if some_condition:
    # do something
    print(a)
elif some_other_condition:
    # some other condition
    print(b)
else:
    # do something else
    print(c)
```

Lecture 7 – part 4

Comparison operators, Chaining comparison operators

Comparison operators (a= 3, b=4)

Operator	Description	Example
`==`	If the values of two operands are equal, then the condition becomes true.	`(a == b) is not true.`
`!=`	If values of two operands are not equal, then condition becomes true.	`(a != b) is true`
`>`	If the value of left operand is greater than the value of right operand, then condition becomes true.	`(a > b) is not true.`
`<`	If the value of left operand is less than the value of right operand, then condition becomes true.	`(a < b) is true.`
`>=`	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	`(a >= b) is not true.`
`<=`	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	`(a <= b) is true.`

Chained Comparisons

Expression Type	Example	Equivalent Boolean Expression	Description
Chained Comparisons	A <= B <= C	A <= B and B <= C	Checks if A is less than/equal to B and B is less than/equal to C.
	X >= Y != Z	X >= Y and Y != Z	Checks if X is greater than/equal to Y and Y is not equal to Z.
and & or	A < B **and** B < C **or** C == D	_	Checks if A<B and B<C, or if C is equal to D.
Using not	**not** (A == B)	A != B	Returns True if A is not equal to B.
	not (A > B **and** C > D)	A <= B **or** C <= D	Checks if A is less than/equal to B or C is less than/equal to D.
Nested Conditions	(A < B **or** C > D) **and** E == F	_	Checks if A<B or C>D, and if E is equal to F.
Chaining with not	**not** A < B < C	**not**(A < B **and** B < C) or A >= B **or** B >= C	Negates the entire chained comparison.
Multiple Operators	A < B < C or D != E **and** **not** F > G	_	A combination of chaining, and, or, and not.

Lecture 7 – part 5

For Loops

for loops

- We can use for loops to execute a block of code for each iteration.
- Many objects in Python are **"iterable",** meaning we can iterate over each element.
- Iterate over every item in a **list**,
- Iterate over every character in a **string**,
- Iterate over every key in a **dictionary.**

for loops

- **Syntax of a for loop:**

```
my_iterable = [1,2,3]
for item in my_iterable:
    print(item)
```

Lecture 7 – part 6

While loops

while loops

- While loops continue to execute a block of code **while** some condition remains **True**.

Syntax of the while loop:

```
while some_condition:
    # Do something
else:
    # Do something different
```

break, continue, pass

break – Breaks out the current closest enclosing loop.

continue – Goes to the top of the closest enclosing loop.

pass – Does nothing at all.