Python Summary Document

Contents

1 Types in Python

Name	**Type**	**Description**	**Examples**
Integer	`int`	Whole numbers	`3, 300, 200`
Floating Points	`float`	Numbers with decimal point	`3.5, 62.4, 25.323`
Strings	`str`	Ordered Sequence of Characters	`"cat", "dog", "hello"`
Lists	`list`	Ordered Sequence of Objects	`[10, "hi", 2.3]`
Dictionaries	`dict`	Unordered *Key:Value* pairs	`{"key1":"value1", "key2":"value2"}`

2 Arithmetic and Boolean Operators

Arithmetic Operator	**Meaning**
`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division
`//`	Integer Division
`%`	Modulus (i.e. remainder)

Boolean Operator	**Meaning**
`x or y`	if x is false, then y, else x
`x and y`	if x is false, then x, else y
`not x`	if x is false, then True, else False
`x == y`	if x is equal to y, then True, else False
`x != y`	if x is not equal to y, then True, else False
`x < y`	if x is less than y, then True, else False
`x <= y`	if x is less than or equal to y, then True, else False
`x > y`	if x is greater than y, then True, else False
`x >= y`	if x is greater than or equal to y, then True, else False
`'item' in list1`	if `'item'` is in `list1`, then True, else False
`'item' not in list1`	if `'item'` is not in `list1`, then True, else False

String Operator	**Description**	**Example**	**Meaning**
`+`	Concatenation	`"hello" + "world"`	`"helloworld"`
`upper()`	Upper-Case	`"hello".upper()`	`"HELLO"`
`lower()`	Upper-Case	`"HELLO".lower()`	`"hello"`
`x[i]`	Get i'th letter of x	`"hello"[1]`	`"e"`
`len`	Length	`len("hello")`	`5`

3 Built-in Data Structures: Lists, Tuples, Sets, Dictionaries

Lists

- Lists store a sequence of mutable items (i.e. you can change its content once created)

Example: List Initialization

```
>>> fruits = ['apple','orange','pear','banana']
>>> fruits[0]
'apple'

```

Example: List Concatenation

```
>>> otherFruits = ['kiwi','strawberry']
>>> fruits + otherFruits
>>> ['apple', 'orange', 'pear', 'banana', 'kiwi', 'strawberry']

```

Example: List Indexing
Note: Python also allows negative-indexing from the back of the list.
For instance, `fruits[-1]` will access the last element `'banana'`.

```
>>> fruits[-2]
'pear'
>>> fruits.pop()
'banana'
>>> fruits
['apple', 'orange', 'pear']
>>> fruits.append('grapefruit')
>>> fruits
['apple', 'orange', 'pear', 'grapefruit']
>>> fruits[-1] = 'pineapple'
>>> fruits
['apple', 'orange', 'pear', 'pineapple']

```

Example: List Slicing
Note: Python also allows us to index multiple adjacent elements at once.
In general `fruits[start:stop]` will get the elements in `start, start+1, ..., stop-1`.
We can also do `fruits[start:]` which returns all elements starting from the `start` index.
Also, `fruits[:end]` will return all elements before the element at position `end`.

```
>>> fruits[0:2]
['apple', 'orange']
>>> fruits[:3]
['apple', 'orange', 'pear']
>>> fruits[2:]
['pear', 'pineapple']
>>> len(fruits)
4

```

Example: Lists of Lists

```
>>> lstOfLsts = [['a','b','c'],[1,2,3],['one','two','three']]
>>> lstOfLsts[1][2]
3
>>> lstOfLsts[0].pop()
'c'
>>> lstOfLsts
[['a', 'b'],[1, 2, 3],['one', 'two', 'three']]

```

Tuples

- A tuple is like a list except that it is immutable once it is created (i.e. you cannot change its content once created)
- tuples are surrounded with parentheses while lists have square brackets

Example

```
>>> pair = (3,5)
>>> pair[0]
3
>>> x,y = pair
>>> x
3
>>> y
5
>>> pair[1] = 6
TypeError: object does not support item assignment

```

Sets

- A set is an unordered list with no duplicate items
- sets are surrounded with curly braces

Example: Set difference, intersection, and union

```
>>> shapes = ['circle','square','triangle','circle']
>>> setOfShapes = set(shapes)
>>> setOfShapes
{'circle', 'square', 'triangle'}
>>> setOfShapes.add('polygon')
>>> setOfShapes
{'polygon', 'circle', 'square', 'triangle'}
>>> 'circle' in setOfShapes
True
>>> 'rhombus' in setOfShapes
False
>>> favoriteShapes = ['circle','triangle','hexagon']
>>> setOfFavoriteShapes = set(favoriteShapes)
>>> setOfShapes - setOfFavoriteShapes
{'polygon', 'square'}
>>> setOfShapes & setOfFavoriteShapes
{'circle', 'triangle'}
>>> setOfShapes | setOfFavoriteShapes
{'polygon', 'square', 'circle', 'hexagon', 'triangle'}

```

Dictionaries

- A dictionary stores a map from one type of object (the key) to another (the value)
- The key must be an immutable type (string, number, or tuple)
- The value can be any Python data type

Example

```
>>> studentIds = {'knuth': 42.0, 'turing': 56.0, 'nash': 92.0 }
>>> studentIds['turing']
56.0
>>> studentIds['nash'] = 'ninety-two'
>>> studentIds
{'knuth': 42.0, 'turing': 56.0, 'nash': 'ninety-two'}
>>> del studentIds['knuth']
>>> studentIds
{'turing': 56.0, 'nash': 'ninety-two'}
>>> studentIds['knuth'] = [42.0,'forty-two']
>>> studentIds
{'knuth': [42.0, 'forty-two'], 'turing': 56.0, 'nash': 'ninety-two'}
>>> studentIds.keys()
dict_keys(['turing', 'nash', 'knuth'])
>>> studentIds.values()
dict_values([56.0, 'ninety-two', [42.0, 'forty-two']])
>>> studentIds.items()
dict_items([('turing', 56.0), ('nash', 'ninety-two'), ('knuth', [42.0, 'forty-two'])])
>>> len(studentIds)
3
```

4 If-Statements

- If-statements are used to test for particular conditions

Example: Simple If-Test 1
Suppose you save your code in `simpleIfTest1.py`.

```python
age = 19

if age >= 18:
    print("you can vote!")

```

If you run `simpleIfTest1.py`, you will get the following output:

```
"you can vote!"

```

Example: Simple If-Test 2
Suppose you save your code in `simpleIfTest2.py`.

```python
age = 15

if age >= 18:
    print("you can vote!")

```

If you run `simpleIfTest2.py`, you will get the following output:

```


```

Note that nothing prints!

Example: If-elif-else 1
Suppose you save your code in `IfElifElse1.py`.

```python
age = 2

# Initialize ticket_price to some random value
ticket_price = 0

if age < 4:
    ticket_price = 0
elif age < 18:
    ticket_price = 10
else:
    ticket_price = 15

print(ticket_price)

```

If you run `IfElifElse1.py`, you will get the following output:

```
0

```

Example: If-elif-else 2
Suppose you save your code in `IfElifElse2.py`.

```python
age = 12

# Initialize ticket_price to some random value
ticket_price = 0

if age < 4:
    ticket_price = 0
elif age < 18:
    ticket_price = 10
else:
    ticket_price = 15

print(ticket_price)

```

If you run `IfElifElse2.py`, you will get the following output:

```
10

```

Example: If-elif-else 3
Suppose you save your code in `IfElifElse3.py`.

```python
age = 20

# Initialize ticket_price to some random value
ticket_price = 0

if age < 4:
    ticket_price = 0
elif age < 18:
    ticket_price = 10
else:
    ticket_price = 15

print(ticket_price)

```

If you run `IfElifElse3.py`, you will get the following output:

```
15

```

5 Loops

Example: For-Loop
Suppose you save your code in `forLoopEx.py`.

```python
for i in range(3):
    print(i)

```

If you run `forLoopEx.py`, you will get the following output:

```
0
1
2

```

Example: While-Loop
Suppose you save your code in `whileLoopEx.py`.

```python
current_value = 0
while current_value <= 5:
    print(current_value)
    current_value += 1

```

If you run `whileLoopEx.py`, you will get the following output:

```
0
1
2
3
4

```

6 Functions

Example: Simple Function
Suppose you save your code in `simpleFunction.py`.

```python
# Display a simple greeting
def greet_user():
    print("hello")

greet_user()

```

If you run `simpleFunction.py`, you will get the following output:

```
hello

```

Example: Passing an Argument
Suppose you save your code in `functionWithArgument.py`.

```python
# Display a personalized greeting
def greet_user(name):
    print("hello ", name + "!")

greet_user("bob")

```

If you run `functionWithArgument.py`, you will get the following output:

```
hello bob!

```

Example: Default Parameter Values
Suppose you save your code in `functionWithDefaultArgument.py`.

```python
# Display a personalized greeting
def greet_user(name, greeting="hello"):
    print(greeting, " " name + "!")

greet_user("bob")
greet_user("bob", "hola")

```

If you run `functionWithDefaultArgument.py`, you will get the following output:

```
hello bob!
hola bob!

```

Example: Returning a value
Suppose you save your code in `functionWithReturnValue.py`.

```
# add two numbers
def add_numbers(x,y):
    return x + y

add_numbers(3,5)

```

If you run `functionWithReturnValue.py`, you will get the following output:

```
8

```

Note, by default, functions return `None` if you don't explicitly return something.

Example: Main Function
Suppose you save your code in `fruit.py`.

```
fruitPrices = {'apples':2.00, 'oranges': 1.50, 'pears': 1.75}

def buyFruit(fruit, numPounds):
    if fruit not in fruitPrices:
        print("Sorry we don't have {}".format(fruit))
    else:
        cost = fruitPrices[fruit] * numPounds
        print("That'll be {} please".format(cost))

# Main Function
if __name__ == '__main__':
    buyFruit('apples',2.4)
    buyFruit('coconuts',2)

```

If you run `fruit.py`, you will get the following output:

```
That'll be 4.800000 please
Sorry we don't have coconuts

```

7 Sources

http://www.cs.cmu.edu/~./15281/assignments/programming/tutorial/index.html#Operators
https://medium.com/@shawnren527/learn-about-python-3-data-types-numbers-and-strings-76c75a917c9b
https://ehmatthes.github.io/pcc/cheatsheets/README.html