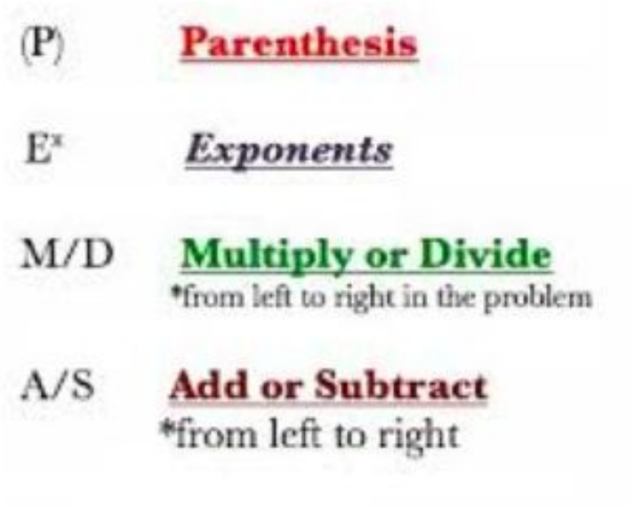
Python basics

Seham Eldeen

Numbers

1. Types of Numbers:
    a. Integers: whole numbers [+ve and -ve]
    b. Float Numbers: have decimal points
2. Basic Arithmetic:
    a. Python works like a calculator
        i. + - / *
        ii. powers/ roots
        iii. order of operations



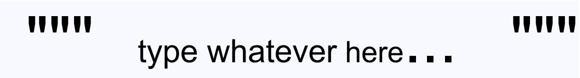
(P)	Parenthesis
E^x	*Exponents*
M/D	Multiply or Divide *from left to right in the problem
A/S	Add or Subtract *from left to right

Dynamic Typing

You don’t need to declare what a variable type is going to be before you do the assignment.

Comments

1. Single Line Comments [using #]
2. Multi-Line Comments



```
"""                              """
        type whatever here...
```

Rules for Variable Name

1. Names cannot start with a number.

    ex: 2made X

2. Names cannot contain spaces, use _ instead.

    ex: two days X, and make it “**two_days**”

3. Names cannot start with symbols **:'",<>/?|\()!@#$%^&*~-+**
4. It's best practice to use names with lowercase.

Strings

- Used to hold text information
- Are indicated with the use of single or double quotes
- Are a sequence of characters

Example:

‘hello’

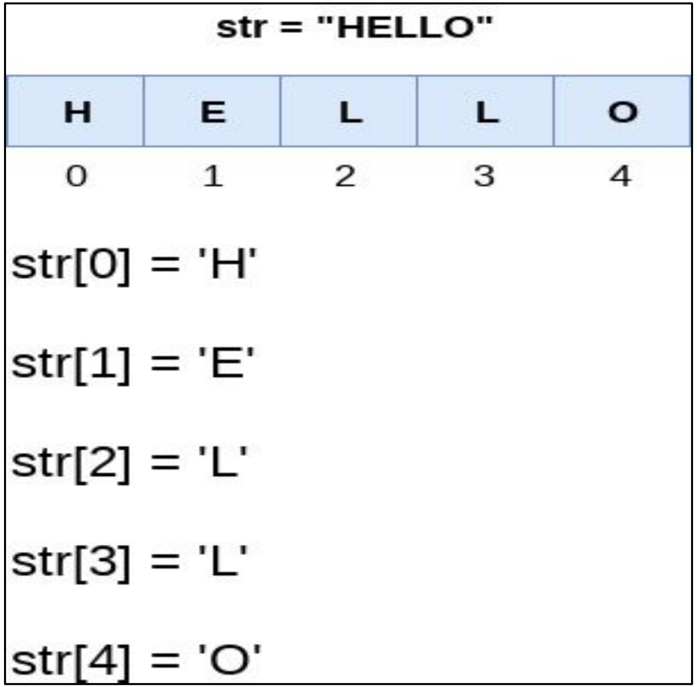
“Hello”

“I’m playing around”   // double quotes to wrap single quotes

String Indexing

[0] first letter

[-1] last letter - Negative Indexing



str = "HELLO"

H	E	L	L	O
0	1	2	3	4

```
str[0] = 'H'

str[1] = 'E'

str[2] = 'L'

str[3] = 'L'

str[4] = 'O'
```

String Slicing

- Has three parts:
  a. Start of the slice
  b. End of the slice
  c. Step size
- **Strings are Immutable:**
  - Means you cannot redefine a particular item assignment index

str[start : End]

Start: start from this element

End: end at this element [exclude this element]

Example:

mystring = ‘abcdefg’

mystring [0] = ‘m’ **X**

String Concatenation: The + operator is used to concatenate strings

```
s = ‘Love’ + ‘Coding’
print(s) // LoveCoding


letter = ‘z-’
print(letter * 10) // z-z-z-z-z-z-z-z-z-z-
```

Built In string methods

1. **Upper:** Returns a copy of the string converted to uppercase
2. **Lower:** Returns a copy of the string converted to lowercase
3. **Capitalize:** Returns a copy of the string with only it’s first letter capitalized
4. **Split:** It splits a string, and also allows you to split on any element of the string

Print formatting

.format(): to add formatted objects to printed string statements

1. **Without variables:**

Syntax: ‘This is a string: { } ’.format(“insert me”)

2. **Defining variables inside of format**

Syntax:

‘This is a string {**var1**} and {**var2**}’.format(**var1** = “something1”, **var2**=”something2”)

Lists: Are python form of Arrays.

examples:

List can contain numbers, strings, nested lists, etc.

```
my_list = [1,2,3]

my_list = [‘strings’, 1, 2, 3.9, True, [1,2,3] ]

print(len(my_list))
```

The len() function will tell you how many items are in the sequence of your list

Indexing & Slicing:

List Indexing:

```
mylist = [‘a’, ‘b’, ‘c’]

print(mylist[0]) // a
print(mylist[1]) // b
print(mylist[2]) // c
print(mylist[-1]) // c
```

List Slicing:

```
mylist = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’]

print(mylist[1:]) // [‘b’, ‘c’, ‘d’, ‘e’]
print(mylist[:3]) // [‘a’, ‘b’, ‘c’]
```

List Concatenation:

```
my_list = ['one','two']
my_list = my_list + ['new item']
print(my_list)  // [‘one’, ‘two’,  ‘new item’]
```

Replicating a list:

```
my_list = ['1', '2'] *2
print(my_list)  // ['1', '2', '1', '2']
```

Unlike strings, lists are Mutable

Means the elements inside a list can be changed

Reassignment:

```
my_list = ['one','two', 'three']
my_list[0] = 'first item change'
print(my_list)  // [‘first item change’, ‘two’, ‘three’]
```

Built In list methods

1. **.append():** permanently add an item to the end of a list
2. **.entend():** extending the original list to include the items of another list
3. **.pop():** to remove (grab) the last item from a list & return it
4. **.reverse():** to reverse the order of your list permanently.
5. **.sort():** to sort the list (alphabetical order) & for numbers, it will order them in ascending order

Nested Lists

One of the main features of python data structures is that they support nesting.

Nesting means having a list inside another list.

```
my_list = [1, 2, ['x', 'y', 'z']]
print(my_list[2]) //['x', 'y', 'z']
print(my_list[2][0]) // x
print(my_list[2][2]) // z
```

Dictionaries

- Allow you to create key-value pairs
- They don’t follow any order because dictionaries follow key-value pair system

Example:

```
my_dict = {
          'key1': 'value1' ,
          'Key2':'value2'   }
```

To grab a value by the key:

```
print(my_dict[‘key1’]) //  value1
print(my_dict[‘key2’]) // value2
```

Dictionaries:

Are flexible with the data types they can hold.

```
my_dict = {  'key1':123,
             'Key2':[12,23,33],
             'key3':['item0','item1','item2']}
print(my_dict)
```

Note: it won't always be printed in order, because dictionaries don't retain any order

Reassigning Dictionary Items:

```
my_dict = {'lunch': 'pizza', 'breakfast': 'eggs'}
print(my_dict['lunch'])// pizza
my_dict['lunch'] = 'burger'
print(my_dict['lunch']) // burger
```

ADD a new key:

```
my_dict['dinner'] = 'pasta'

print(my_dict)//{'lunch': 'burger', 'breakfast': 'eggs', 'dinner': 'pasta'}
```

Dictionary Methods:

.keys(): to return a list of all the keys

.values(): to return a list of all the values

.items(): to return tuples of all items

```
d = {'key1':1,'key2':2,'key3':3}
print (d.keys()) //['key1', 'key2', 'key3']
print (d.values()) //[1, 2, 3]
print(d.items())//[('key1', 1), ('key2', 2), ('key3', 3)]
```

Tuples

- Similar to lists, except you can’t index a tuple and try to change it
- They are immutable, meaning they cannot change
- You would use tuples to present stuff that cannot be changes such as weekdays, calendar days etc.

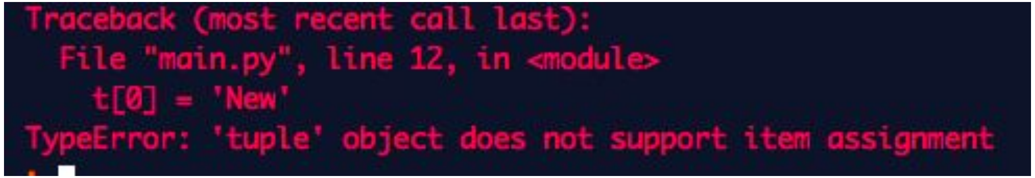
<u>Creating a tuple</u>: using ( ) with elements separated by a comma/ tuples can hold mixed data types

```
t = ('a', True, 1, 12)

Print(t) // ('a', True, 1, 12)

print(t[0]) // ‘a’

t[0] = 'New'
```



```
Traceback (most recent call last):
  File "main.py", line 12, in <module>
    t[0] = 'New'
TypeError: 'tuple' object does not support item assignment
```

Basic Tuple Methods

.index(): to enter a value and return the index of that value

.count(): to count the number of times a value appears

```
t = ('a', True, 1, 12, 'a')
print(t.index(12)) // 3
print(t.count('a')) // 2
```

Sets

- Are an unordered collection of **unique** elements

- <u>**Creating a set:**</u> using the Set() function

- You can also convert a list into a set

```
x = set()
x.add(1)
x.add(2)
x.add(3)
x.add(4)
x.add(4)
x.add(4)
x.add(5)
print(x) //{1,2,3,4,5}
```

Exercise 1

Given the string:

s = 'django'

Use indexing to print out the following:

- 'd'
- 'O'
- 'Djan'
- 'Jan'
- 'go'

Then use indexing to reverse the string

Exercise 2

Given this nested list:

```
l = [3,7,[1,4,'hello']]
```

Reassign "hello" to be "goodbye"

Exercise 3

Using keys and indexing, grab the 'hello' from the following dictionaries:

```
d1 = {'simple_key':'hello'}

d2 = {'k1': {'k2':'hello'} }

d3 = { 'k1':[ {'nest_key':['this is deep',['hello'] ] } ] }
```

Exercise 4

Use a set to find the unique values of the list below:

```
mylist = [1,1,1,1,1,2,2,2,2,3,3,3,3]
```

Exercise 5

You are given two variables:

```
age = 21
name = "Seham"
```

Use print formatting to print the following string:

"Hello my name is Seham and I’m 21 years old"