Conditionals on strings, ASCII, UNICODE Character sets, Built_in functions

January 25, 2023

1 Single line if else statements (Kind of ternary)

Syntax:
expression1 if condition else expression2

[5]:
```
n = int(input())
if n % 2 == 0:
    print('Even')
else:
    print('Odd')
```

```
15
Odd
```

[6]:
```
n = int(input())
print('Even') if n % 2 == 0 else print('Odd')
# print('Even') --> Expression 1
# n % 2 == 0 --> Condition
# print('Odd') --> Expresssion 2
```

```
15
Odd
```

[7]:
```
a, b = map(int, input().split())
if a > b:
    largest = a
else:
    largest = b
print(largest)
```

```
10 20
20
```

[8]:
```
a, b = map(int, input().split())
largest = a if a > b else b
print(largest)
```

```
10 20
20
```

2 continued single line if else statements

Syntax:
expression1 if condition1 else expression2 if condition2 else expression3 if condition3 ..... else expression n

[9]:
```
# largest of three distinct numbers
a, b, c = map(int, input().split())
if a > b and a > c:
    print(a)
elif b > a and b > c:
    print(b)
else:
    print(c)
```

```
10 20 30
30
```

[10]:
```
# largest of three distinct numbers
a, b, c = map(int, input().split())
print(a) if a > b and a > c else print(b) if b > a and b > c else print(c)
```

```
10 20 30
30
```

[11]:
```
# largest of three distinct numbers
a, b, c = map(int, input().split())
largest = a if a > b and a > c else b if b > a and b > c else c
print(largest)
```

```
10 20 30
30
```

3 Conditionals on Strings

[12]:
```
if 'ironman' > 'thor':
    print('ironman is powerful than thor')
else:
    print('thor is powerful than ironman')
```

```
thor is power is than ironman
```

[13]:
```
if 'a' > 'A':
    print('a is greater than A')
else:
```

```
    print('A is greater than a')
```

```
a is greater than A
```

4 ASCII Character Set

- 1963
- ASCII Character Set
- American Standard Code for Information Interchange
- Latin Alphabets (a-z, A-Z)
- Digits (0-9)
- Special Characters (~%$#@!)
- 128 character
- CODE POINT VALUE to each character (Number for each character)
- a-z –> 97-122
- A-Z –> 65-90
- 0-9 –> 48-57
- ' ' –> 32
- chr()
- ord()

4.1 ord()

- ord() takes a character and returns it's CODE POINT value.

[14]:
```
print(ord('a'))
```

```
97
```

[15]:
```
print(ord('A'))
```

```
65
```

[16]:
```
if 'a' > 'A': # 97 > 65
    print('a is greater than A')
else:
    print('A is greater than a')
```

```
a is greater than A
```

[18]:
```
if 'ironman' > 'thor': # i > t --> 105 > 118 False
    print('ironman is powerful than thor')
else:
    print('thor is powerful than ironman')
```

```
thor is powerful than ironman
```

[20]:
```
if 'ironman' > 'Thor': # i > T --> 105 > 80 True
    print('ironman is powerful than thor')
else:
```

```
    print('thor is powerful than ironman')
```

```
ironman is powerful than thor
```

[21]:
```
print(ord(' '))
```

```
32
```

[22]:
```
print(ord('@'))
```

```
64
```

4.2 chr()

- chr() takes a number. And it will return the character associated with that number in ASCII character set

[23]:
```
print(chr(97))
```

```
a
```

[24]:
```
print(chr(118))
```

```
v
```

[25]:
```
print(chr(64))
```

```
@
```

5 UNICODE

- First version of UNICODE is published in 1991.
- ASCII 128 character
- UNICODE 144000 characters
- UNICODE is having 159 scripts (natural languages)
- UNICODE also supports emojies

[ ]:
```
# English --> 65-90, 97-122
# Telugu --> 3072, 3200
```

[26]:
```
print(chr(3077))
```

[27]:
```
print(chr(3080))
```

[28]:
```
for i in range(3072, 3201):
    print(i, '-->', chr(i))
```

3072 -->
3073 -->
3074 -->
3075 -->
3076 -->
3077 -->
3078 -->
3079 -->
3080 -->
3081 -->
3082 -->
3083 -->
3084 -->
3085 -->
3086 -->
3087 -->
3088 -->
3089 -->
3090 -->
3091 -->
3092 -->
3093 -->
3094 -->
3095 -->
3096 -->
3097 -->
3098 -->
3099 -->
3100 -->
3101 -->
3102 -->
3103 -->
3104 -->
3105 -->
3106 -->
3107 -->
3108 -->
3109 -->
3110 -->
3111 -->
3112 -->
3113 -->
3114 -->
3115 -->
3116 -->
3117 -->
3118 -->
3119 -->

3120 -->
3121 -->
3122 -->
3123 -->
3124 -->
3125 -->
3126 -->
3127 -->
3128 -->
3129 -->
3130 -->
3131 -->
3132 -->
3133 -->
3134 -->
3135 -->
3136 -->
3137 -->
3138 -->
3139 -->
3140 -->
3141 -->
3142 -->
3143 -->
3144 -->
3145 -->
3146 -->
3147 -->
3148 -->
3149 -->
3150 -->
3151 -->
3152 -->
3153 -->
3154 -->
3155 -->
3156 -->
3157 -->
3158 -->
3159 -->
3160 -->
3161 -->
3162 -->
3163 -->
3164 -->
3165 -->
3166 -->
3167 -->

```
3168 -->
3169 -->
3170 -->
3171 -->
3172 -->
3173 -->
3174 -->
3175 -->
3176 -->
3177 -->
3178 -->
3179 -->
3180 -->
3181 -->
3182 -->
3183 -->
3184 -->
3185 -->
3186 -->
3187 -->
3188 -->
3189 -->
3190 -->
3191 -->
3192 -->
3193 -->
3194 -->
3195 -->
3196 -->
3197 -->
3198 -->
3199 -->
3200 -->
```

[30]:
```python
# Print my name in native language
name = chr(3114) + chr(3125) + chr(3112) + chr(3149)
print(name)
```

[31]:
```python
# 128512
for i in range(128512, 128700):
    print(chr(i), end = ' ')
```

[32]:
```
if ' ' > 'a': #
    print('YES')
else:
    print('NO')
```

```
YES
```

[35]:
```
# string1 is greater than string2

if 'amit' > 'aman': # i > a
    print('YES')
else:
    print('NO')
```

```
YES
```

5.1 String comparison

- It compares two string until the first differing characters encountered.
- string1 is greater than string2 –> string2 should be before string1
- string1 is less than string2 –> string1 should be before string2 when you arrange the strings alphabetically

[36]:
```
names = ['amit', 'arun', 'amala', 'aman']
print(names)
# sort() --> sorts in ascending order
names.sort()
print(names)
```

```
['amit', 'arun', 'amala', 'aman']
['amala', 'aman', 'amit', 'arun']
```

6 Built_in functions in Python

- 70 Built_in functions are there in Python (as of Python 3.10.6)

A	B	C	D	E	F	G	H	I	L
abs()	bin()	callable()	delattr()	enumerate()	filter()	getattr()	hasattr()	id()	len()
aiter()	bool()	chr()	dict()	eval()	float()	globals()	hash()	input()	list()
all()	breakpoint()	classmethod()	dir()	exec()	format()		help()	int()	locals()
any()	bytearray()	compile()	divmod()		frozenset()		hex()	isinstance()	
anext()	bytes()	complex()						issubclass()	
ascii()								iter()	

M	N	O	P	R	S	T	V	Z
map()	next()	object()	pow()	range()	set()	tuple()	vars()	zip()
max()		oct()	print()	repr()	setattr()	type()		

M	N	O	P	R	S	T	V	Z
memoryview() min()		open() ord()	property()	reversed() round()	slice() sorted() staticmethod() str() sum() super()			

- Built_in functions are used to perform a specific task, which would take a few lines of code if the function doesn't exist.

[37]:
```
# Find out the minimum of two numbers
a, b = map(int, input().split())
if a < b:
    minimum = a
else:
    minimum = b
print(minimum)
```

```
5 6
5
```

6.1 min()

- smallest of n values

[38]:
```
a, b = map(int, input().split())
print(min(a, b))
```

```
5 6
5
```

[39]:
```
a, b, c = map(int, input().split())
print(min(a, b, c))
```

```
10 20 30
10
```

6.2 abs()

- Returns the absolute value
- Positive value

[40]:
```
print(abs(10))
```

```
10
```

[41]:
```
print(abs(-10))
```

```
10
```

[43]:
```
a = 10
b = 20
print(abs(a - b))
```

```
10
```

[44]:
```
a = 10
b = 20
if a > b:
    print(a - b)
else:
    print(b - a)
```

```
10
```

6.3 Truthiness of values in Python

- Every value that belongs to a specific type can be checked for their trutiness in Python.
- Every numeric type is treated as True except 0, 0.0
- Every sequence type is treated as True, if and only if they contain atleast one element inside them
- Empty sequences will be treated as False

[45]:
```
if True: # 10 > 20 --> False
    print('yes')
else:
    print('no')
```

```
no
```

[46]:
```
if 10: # 10 --> True
    print('if block is executed')
else:
    print('else block is executed')
```

```
if block is executed
```

[47]:
```
if -10:
    print('if block is executed')
else:
    print('else block is executed')
```

```
if block is executed
```

[48]:
```
if 0:
    print('if block is executed')
else:
    print('else block is executed')
```

```
else block is executed
```

[49]:
```
if 2.2:
    print('if block is executed')
else:
    print('else block is executed')
```

```
if block is executed
```

[50]:
```
10 --> True or False
[10, 20, 30, 40]
```

[50]:
```
False
```

6.4 bool()

- Takes a value and returns either True or False

[51]:
```
print(bool(12))
```

```
True
```

[52]:
```
print(bool(-12))
```

```
True
```

[53]:
```
print(bool(123.456))
```

```
True
```

[54]:
```
print(bool(-123.456))
```

```
True
```

[55]:
```
print(bool(0))
```

```
False
```

[56]:
```
print(bool(0.0000000))
```

```
False
```

[57]:
```
print(bool(0.00001))
```

```
True
```

[58]:
```
if [[]]: # True
    print('YES')
else:
    print('NO')
```

```
YES
```

[59]:
```
print(bool([10, 20, 30]))
```

```
True
```

[60]:
```
print(bool([]))
```

```
False
```

[61]:
```
t = (10)
print(bool(t))
```

```
True
```

[62]:
```
t = ()
print(bool(t))
```

```
False
```

[63]:
```
s = 'hello'
print(bool(s))
```

```
True
```

[64]:
```
s1 = ''
print(bool(s1))
```

```
False
```

[65]:
```
s1 = ' '
print(bool(s1))
```

```
True
```

[66]:
```
if 'hello':
    print('yes')
else:
    print('no')
```

```
yes
```

[67]:
```
if '':
    print('yes')
else:
    print('no')
```

```
no
```

6.5 all()

- we can only apply all() function on iterables (lists, tuples, sets, strings, range)
- True will be returned if all elements in the iterable are True, otherwise it will return False

[68]:
```
lst = [10, 20, 30, 40]
print(all(lst))
```

```
True
```

[69]:
```
lst = [10, 20, 0, 40]
print(all(lst))
```

```
False
```

[70]:
```
lst = [(), (), (), ()]
print(all(lst))
```

```
False
```

[71]:
```
lst = [(10), (29), (35), ()]
print(all(lst))
```

```
False
```

[72]:
```
lst_of_strings = [' ', ' ', ' ', ' ']
print(all(lst_of_strings))
```

```
True
```

[73]:
```
print(all(123))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [73], in <cell line: 1>()
----> 1 print(all(123))

TypeError: 'int' object is not iterable
```

6.6 help()

- help(thethingthatyouwanttogethelp)

[74]:
```
help(all)
```

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

all(iterable, /)
    Return True if bool(x) is True for all values x in the iterable.

    If the iterable is empty, return True.
```

6.7 any()

- Returns True, even if one of the values in the iterable is True, else returns False

[75]:
```
lst = [10, 20, 30, 40]
print(any(lst))
```

```
True
```

[76]:
```
lst = [0.0, 0, 0, 40]
print(any(lst))
```

```
True
```

[77]:
```
lst = ['', [], (), 2.2]
print(any(lst))
```

```
True
```

[78]:
```
lst = ['', [], (), 0.0]
print(any(lst))
```

```
False
```

[79]:
```
print(any(123))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [79], in <cell line: 1>()
----> 1 print(any(123))

TypeError: 'int' object is not iterable
```

6.8 Explicit Type Conversion

6.9 int()

- Converts a floating point value or a string into an integer
- String should contain digits only

[2]:
```
f = 12.2
i = int(f)
print(i)
print(type(i))
```

```
12
<class 'int'>
```

[3]:
```
s = '123'
i = int(s)
print(i)
print(type(i))
```

```
123
<class 'int'>
```

[4]:
```
s = 'abc'
i = int(s)
print(i)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [4], in <cell line: 2>()
      1 s = 'abc'
----> 2 i = int(s)
      3 print(i)

ValueError: invalid literal for int() with base 10: 'abc'
```

[5]:
```
print(int('101')) # Decimal value (Base 10)
```

```
101
```

[6]:
```
print(int('101', 2)) # Decimal value (Base 2)
```

```
5
```

[7]:
```
bin(5)
```

[7]: '0b101'

[9]:
```
int('10101')
```

[9]: 10101

[11]:
```
int('1010101', 2)
```

[11]: 85

[12]:
```
int('1010101')
```

[12]: 1010101

6.10 float()

- It converts an integer or a string into a floating point value
- String should contain only digits

[13]:
```
i = 10
f = float(i)
print(f)
print(type(f))
```

```
10.0
<class 'float'>
```

[14]:
```
s = '123'
f = float(s)
print(f)
print(type(f))
```

```
123.0
<class 'float'>
```

[15]:
```
s = '123abc'
f = float(s)
print(f)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [15], in <cell line: 2>()
      1 s = '123abc'
----> 2 f = float(s)
      3 print(f)

ValueError: could not convert string to float: '123abc'
```

[ ]:
```
a = int(input()) # Type conversion
```

[18]:
```
a = int(input())
print(a)
print(type(a))
```

```
10
10
<class 'int'>
```

6.11 str()

- Converts anything into string

[19]:
```
a = 10
s = str(a)
print(s)
print(type(s))
```

```
10
<class 'str'>
```

[21]:
```
a = 12.34
s = str(a)
```

```
print(s * 3)
print(type(s))
```

```
12.3412.3412.34
<class 'str'>
```

[23]:
```
lst = [10, 20, 30]
s = str(lst)
print(s[0])
print(s)
```

```
[
[10, 20, 30]
```

[25]:
```
x = 123456
s = str(x)
print(len(s))
```

```
6
```

6.12 list()

- Converts any other sequence type or a map object into a list

[29]:
```
t = (20, 30, 10) # 10 20 30
l = list(t)
l.sort()
print(l)
print(type(l))
```

```
[10, 20, 30]
<class 'list'>
```

[31]:
```
s = 'hello'
l = list(s)
print(l)
```

```
['h', 'e', 'l', 'l', 'o']
```

[33]:
```
s = 'hallo' # Strings are immutable in Python
    #01234
s[1] = 'e'
print(s)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [33], in <cell line: 3>()
      1 s = 'hallo'
      2     #01234
----> 3 s[1] = 'e'
```

```
      4 print(s)

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

[38]:
```
s = 'hallo'
l = list(s) # In python lists are mutable
print(l)
l[1] = 'e'
print(l)
print(''.join(l))
```

```
['h', 'a', 'l', 'l', 'o']
['h', 'e', 'l', 'l', 'o']
hello
```

[35]:
```
l = [10, 20, 30]
#    0   1   2
print(l)
l[2] = 40
print(l)
```

```
[10, 20, 30]
[10, 20, 40]
```

[41]:
```
s = 'rolex'
# output --> elorx
l = list(s)
print(l)
l.sort()
print(l)
print(''.join(l))
```

```
['r', 'o', 'l', 'e', 'x']
['e', 'l', 'o', 'r', 'x']
elorx
```

[42]:
```
n = 123 # Numeric types - int, float (not iterables)
l = list(n) # iterable list, set, string, tuple, range()
print(l)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [42], in <cell line: 2>()
      1 n = 123
----> 2 l = list(n)
      3 print(l)
```

```
TypeError: 'int' object is not iterable
```

[43]:
```
# range() into a list
print(list(range(2, 101, 2)))
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42,
44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82,
84, 86, 88, 90, 92, 94, 96, 98, 100]
```

[44]:
```
r = range(2, 101, 2)
l = list(r)
print(l)
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42,
44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82,
84, 86, 88, 90, 92, 94, 96, 98, 100]
```

6.13 tuple()

- Converts any other sequence type into a tuple

[45]:
```
l = [10, 20, 30] # list
t = tuple(l)
print(t)
print(type(t))
```

```
(10, 20, 30)
<class 'tuple'>
```

[46]:
```
s = 'hello'
t = tuple(s)
print(t)
print(type(t))
```

```
('h', 'e', 'l', 'l', 'o')
<class 'tuple'>
```

6.14 set()

- Converts any other sequence type into set
- Property of set - Set will not contain any duplicate elements

[47]:
```
l = [10, 20, 30, 40, 30, 20, 10] # 10 20 30 40
s = set(l)
print(s)
```

```
{40, 10, 20, 30}
```

[48]:
```
s = 'aaaaaaaaaaaaa'
x = set(s)
```

```
print(x)
```

```
{'a'}
```

6.14.1 Pangram

- You'll be given a string of lowercase latin alphabets and spaces
  find out if it's a Pangram or not
  a quick brown fox jumps over the lazy dog

[55]:
```
s = input("Enter a string: ")
x = set(s)
if len(x) == 27:
    print('Pangram')
else:
    print('Not a Pangram')
```

```
Enter a string: a quick brown fox jumps over the lazy dog
Pangram
```

[56]:
```
# Intersection of two sets
s1 = {10, 20, 40, 30}
s2 = {100, 60, 30, 40}
print(s1.intersection(s2))
```

```
{40, 30}
```

[63]:
```
# You'll be given two strings find out how many letters are present
# in both strings (strings will not contain duplicate letters)
s1 = 'liger'
s2 = 'lion'
# output --> 2
x1 = set(s1)
x2 = set(s2)
print(x1, x2)
print(len(x1.intersection(x2)))
```

```
{'g', 'e', 'i', 'r', 'l'} {'n', 'l', 'i', 'o'}
2
```

6.15 len()

- Returns the length of a sequence type (no. of elements)

[49]:
```
a = [10, 20, 50, 70] # length of a
print(len(a))
```

```
4
```

[50]:
```
s = 'this is python'
print(len(s))
```

```
14
```

[52]:
```
n = 123
s = str(n)
print(len(s))
```

```
3
```

6.16 dict()

- Converts some specific type of sequence type into dictionary
- Dictionary contains word: defintion
- movie: director
- actor: no.of movies
- batsment: runs
- bowler: no of hat-tricks

[64]:
```
l = [10, 20, 30]
d = dict(l)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [64], in <cell line: 2>()
      1 l = [10, 20, 30]
----> 2 d = dict(l)

TypeError: cannot convert dictionary update sequence element #0 to a sequence
```

[65]:
```
lst = [('apple', 5), ('orange', 6), ('kiwi', 4)]
d = dict(lst)
print(d)
print(type(d))
```

```
{'apple': 5, 'orange': 6, 'kiwi': 4}
<class 'dict'>
```

[68]:
```
lst = [['apple', 5], ['orange', 6], ['kiwi', 4]] # list of lists
d = dict(lst)
print(d)
print(type(d))
```

```
{'apple': 5, 'orange': 6, 'kiwi': 4}
<class 'dict'>
```

[69]:
```
words = ['az', 'by', 'cx']
d = dict(words)
print(d)
```

```
{'a': 'z', 'b': 'y', 'c': 'x'}
```

[70]:
```
words = ['azp', 'byq', 'cxr']
d = dict(words)
print(d)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [70], in <cell line: 2>()
      1 words = ['azp', 'byq', 'cxr']
----> 2 d = dict(words)
      3 print(d)

ValueError: dictionary update sequence element #0 has length 3; 2 is required
```

[71]:
```
lst = [['apple', 5, 5], ['orange', 6, 6], ['kiwi', 4, 4]] # list of lists
d = dict(lst)
print(d)
print(type(d))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [71], in <cell line: 2>()
      1 lst = [['apple', 5, 5], ['orange', 6, 6], ['kiwi', 4, 4]] # list of lists
----> 2 d = dict(lst)
      3 print(d)
      4 print(type(d))

ValueError: dictionary update sequence element #0 has length 3; 2 is required
```

6.17 max()

- Returns the maximum of a given sequence of elements
- Can be used in two ways
- max(a, b, c, ..., n)
- max(sequence)

[73]:
```
a, b, c, d = 10, 20, 30, 40
print(max(a, b, c, d))
```

```
40
```

[74]:
```
l = [-1, -1.7, -12, -16]
print(max(l))
```

```
-1
```

[75]:
```
s = 'hello'
print(max(s))
```

```
o
```

[76]:
```
s = 'ABCzA' # A-Z --> 65-90, a-z --> 97-122
print(max(s))
```

```
z
```

[78]:
```
lst_of_strings = ['Tokyo', 'nairobi', 'denver', 'berlin', 'professor']
print(max(lst_of_strings)) # T, n, d, b, p
```

```
professor
```

6.18 min()

- Returns the minimum of a given sequence of elements
- Can be used in two ways
- min(a, b, c, ..., n)
- min(sequence)

[79]:
```
a, b, c, d = 10, 20, 30, 40
print(min(a, b, c, d))
```

```
10
```

[80]:
```
l = [-1, -1.7, -12, -16]
print(min(l))
```

```
-16
```

[81]:
```
s = 'hello'
print(min(s))
```

```
e
```

[82]:
```
s = 'ABCzA' # A-Z --> 65-90, a-z --> 97-122
print(min(s))
```

```
A
```

[83]:
```
lst_of_strings = ['tokyo', 'nairobi', 'denver', 'berlin', 'professor']
print(min(lst_of_strings)) # T, n, d, b, p
```

```
berlin
```

6.19 sum()

- Returns the sum of sequences of integer type

[84]:
```
lst = [10, 20, 30]
print(sum(lst))
```

```
60
```

[85]:
```
print(sum(range(1, 11)))
```

```
55
```

[87]:
```
# Sum of first n even natural numbers
# n = 5
# 2 4 6 8 10
n = 5 # 11
sum(range(2, n*2 + 1, 2))
```

[87]:
```
30
```

[88]:
```
# Find out the average of  the list of integers
lst = [10, 2, 11, 9]
# avg =
print(sum(lst) / len(lst))
```

```
8.0
```