Python’s Builtin Functions

The most commonly used Python builtin function: print()

“print()” function displays the value of argument passed to it.

print(5): prints 5

print(“hello world”): prints the message “hello world”

Some other built-ins that come in handy are: input(), len(), type() etc.

first_name = **input**(“Enter your first name: ”)

len(“Rakesh”) --> 6

len()

It tells you the number of characters in the string.

```
>>> len("rakesh")
6

>>> len("vikash")
6
```

It is also able to tell you the number of elements in a list / sequence.

```
>>> len([1, 2, 3, 5, 7])
5

>>> len(['rakesh', 'vikash', 'ashish'])
3
```

Some Numbers Related Builtin Functions

- abs() Returns the absolute value of a number
- bin() Returns the binary version of a number
- complex() Returns a complex number
- divmod() Returns the quotient and the remainder when argument1 is divided by argument2
- max() Returns the largest item in an iterable
- min() Returns the smallest item in an iterable
- pow() Returns the value of x to the power of y
- range() Returns a sequence of numbers, starting from 0 and increments by 1 (by default)
- round() Rounds a number
- sum() Sums the items of an iterator

```
>>> l = [5, 6, 2, 0, 9]
>>> max(l)
9
>>> min(l)
0
>>> sum(l)
22
```

```
>>> round(5.2)
5
>>> round(5.9)
6
>>> round(5)
5
```

range()

Range takes three arguments: start, stop, step
But start and step are optional. What does that mean?
They can some default values, like:
Start: 0
Step: 1
Note: Range is exclusive of ‘stop’

```
>>> list(range(1, 11))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>>
>>>
>>> list(range(2, 20, 2))
[2, 4, 6, 8, 10, 12, 14, 16, 18]
>>>


>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
>>>
>>> list(range(0, 10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
>>> list(range(0, 10, 2))
[0, 2, 4, 6, 8]
>>>
>>> list(range(0, 10, 3))
[0, 3, 6, 9]
>>>
```

pow()

```
>>> pow(2, 5)
32
>>> 2 ** 5
32
```

Python is very flexible. In most situations, it gives you more than one way to do things.

Some Examples

```
x = abs(-7.25)
print(x)
```

```
7.25
```

```
x = bin(36)
print(x)

# The result
will always
have the prefix
0b
```

```
0b100100
```

A	B	C
1	0 * 2^5	32
0	0 * 2^4	0
0	0 * 2^3	0
1	1 * 2^2	4
0	0 * 2^1	0
0	0 * 2^0	0
		36

```
x = complex(3, 5)
print(x)
```

```
(3+5j)
```

```
x = divmod(5, 2)

print(x)
```

```
(2, 1)
```

First: quotient
Second: remainder

```
a = (1, 5, 3, 9)
x = max(a)

print(x)
```

```
9
```

```
x = max("Mike", "John", "Vicky")

print(x)
```

```
Vicky
```

```
x = pow(4, 3)

print(x)
```

```
64
```

Return the value of 4 to the power of 3 (same as 4 * 4 * 4):

ord() and chr()

Here, ord() and chr() are built-ins.

Ord(): gives you ASCII for the alphabet

Chr(): gives you the alphabet for the ASCII

```
>>> ord('A')
65

>>> chr(65)
'A'
```

Some Type Casting Related Builtin Functions

- bool() Returns the boolean value of the specified object
- dict() Returns a dictionary (key-value pairs)
- float() Returns a floating point number
- int() Returns an integer number
- list() Returns a list
- set() Returns a new set object
- str() Returns a string object
- tuple() Returns a tuple (tuple is an immutable list)
- type() Returns the type of an object

Type Casting Built-ins: bool()

```
>>> 0
0
>>> True
True
>>> bool(0)
False
>>> bool(-1)
True
>>> bool(1)
True
>>>
>>> bool('A')
True
>>>
```

Extra Information:
Q1: Trick question about a set.

```
>>> {True, False, True}
{False, True}
>>> {True, False, 0, 1}
{False, True}
```

Q2: Trick question about if-else:

```
var = 0

if(True):
    print("Got True")

if(var):
    print("In If")
else:
    print("In Else")
```

Type Casting Built-ins: bool()

```
if(True):
    print("Got True")
```

This is equivalent to directly saying:
print(“Got True”)

```
if(False):
       print(“Got False”)
```

This is equivalent to not writing any code.
if(False) --> The conditional expression always to evaluates to False

Problem:

```
if(True):
    print("in if")
else:
    print("in else")
```

```
if(False):
    print("in if")
else:
    print("in else")
```

int() and list()

```
>>> int("5")
5
>>> int("5XYZ")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '5XYZ'

>>> age = input("Enter your age: ")
Enter your age: 20
>>> type(age)
<class 'str'>

>>> salary = int(input("Enter your salary: "))
Enter your salary: 5000
>>> type(salary)
<class 'int'>

>>> age
'20'
>>> salary
5000
```

```
>>> float('5.2')
5.2
>>> float('5.2XYZABC')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: could not convert string to float:
'5.2XYZABC'
>>> float(5)
5.0
```

Some Examples of Type Casting Related Builtin Functions

We saw this in code for building a calculator:

num1 = float(input("Enter the 1st num:"))

Use of list() when you want to reverse a list:

```
l = [1, 2, 3, 4, 5]
```

```
reversed(l)
```

```
<list_reverseiterator at 0x7f67747bc6a0>
```

```
list(reversed(l))
```

```
[5, 4, 3, 2, 1]
```

Finding volume of a cube

```
# Variable declaration. s is holding the side length.
s = input("Enter the side: ") # input() gives you a string
print(type(s))
s = float(s) # Type casting builtin for getting the float value of r
volume=pow (s,3) # Computation
print(volume)
```

Some Builtins For Processing a List

- enumerate() Takes a collection (e.g. a tuple) and returns it as an enumerate object
- filter() Use a filter function to exclude items in an iterable object
- iter() Returns an iterator object
- map() Returns the specified iterator with the specified function applied to each item
- range() Returns a sequence of numbers, starting from 0 and increments by 1 (by default)
- reversed() Returns a reversed iterator
- sorted() Returns a sorted list
- slice() Returns a slice object
- zip() Returns an iterator, from two or more iterators

Some Examples of Builtins For Processing a List (Part 1)

```
>>> l = [5, 1, 9, 2, 4, 7] # Here, your list is a sequence of integers.
>>> max(l)
9
>>> min(l)
1
>>> sum(l)
28

>>> sorted(l)
[1, 2, 4, 5, 7, 9]
>>> sorted(l, reverse=True)
[9, 7, 5, 4, 2, 1]

>>> l
[5, 1, 9, 2, 4, 7]
>>>
>>> list(reversed(l))
[7, 4, 2, 9, 1, 5]
>>>
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
>>> list(range(0,10,1))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(0,10,2))
[0, 2, 4, 6, 8]
```

```
>>> l = ["alpha", "beta", "gamma", "delta", "epsilon"] # sequence
of strings
>>> max(l)
'gamma'
>>>
>>> ord('a')
97
>>> ord('b')
98
>>> ord('g')
103
>>> min(l)
'alpha'
>>>
>>> sorted(l)
['alpha', 'beta', 'delta', 'epsilon', 'gamma']
>>> list(reversed(l))
['epsilon', 'delta', 'gamma', 'beta', 'alpha']
>>>
```

Some Examples of Builtins For Processing a List (Part 2)

```
x = ('apple', 'banana', 'cherry')
y = enumerate(x)

print(list(y))
```

```
[(0, 'apple'), (1, 'banana'), (2, 'cherry')]
```

```
ages = [5, 12, 17, 18, 24, 32]

def myFunc(x):
  if x < 18:
    return False
  else:
    return True

adults = filter(myFunc, ages)

for x in adults:
  print(x)
```

```
18
24
32
```

```
x = iter(["apple", "banana", "cherry"])
print(next(x))
print(next(x))
print(next(x))
```

```
apple
banana
cherry
```

```
def myfunc(a):
  return len(a)

x = map(myfunc, ('apple', 'banana', 'cherry'))

print(x)

#convert the map into a list, for readability:
print(list(x))
```

```
<map object at 0x056D44F0>
[5, 6, 6]
```

```
x = range(6)

for n in x:
  print(n)
```

```
0
1
2
3
4
5
```

```
a = ("John", "Charles", "Mike")
b = ("Jenny", "Christy", "Monica")

x = zip(a, b)

#use the tuple() function to
display a readable version of the
result:

print(tuple(x))
```

```
(('John', 'Jenny'), ('Charles', 'Christy'), ('Mike', 'Monica'))
```

Practice Problems

1. Create a dictionary using Python built-ins that would have alphabets as keys and ASCII code as values.

2. Create a dictionary using Python built-ins that would have ASCII code as the key, and alphabets as values.

Ref: https://www.w3schools.com/python/python_ref_functions.asp