# Programming and Data Structures with Python

# Lecture 06, 04 January 2021 ¶

## Functions and parameters

- Pass a mutable value, then it can updated in the function
- Immutable values will be copied

## Name spaces

- Variables within functions are local
- But you can use a global immutable value

In [1]:

```
def factorial(n):
    fact = 1
    while (n > 0):
        fact = fact * n
        n = n -1
    return(fact)
```

In [12]:

```
x = 6
factorial(x)
```

Out[12]:

```
720
```

In [13]:

```
x
```

Out[13]:

```
6
```

In [8]:

```
def factorial2(n):
    i = 1      # This i is local to the function
    fact = 1
    while (i <= n):
        fact = fact * i
        i = i+1
    return(fact)
```

In [10]:

```
l = []
for i in range(10):
    l.append(factorial2(i))
```

In [11]:

```
l
```

Out[11]:

```
[1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880]
```

In [15]:

```
def g(n):
    l.append(n)  # There is no l defined inside g(), so this is a global

l = [4]
g(6)
```

In [16]:

```
l
```

Out[16]:

```
[4, 6]
```

In [18]:

```
def h(n):
    j = i
    return(j)

i = 7
h(i)
```

Out[18]:

```
7
```

## Strings

- Text
- Sequence of characters, operations are similar to a list
- But immutable
- Denote a string using single, double or triple quotes
- No separate single character type; a single character is a string of length 1

In [23]:

```
s = "hello"
t = ' there'
```

In [24]:

```
s+t
```

Out[24]:

```
'hello there'
```

In [22]:

```
s[1]
```

Out[22]:

```
'e'
```

In [25]:

```
book="Madhavan's book"
```

In [26]:

```
book
```

Out[26]:

```
"Madhavan's book"
```

In [27]:

```
quote='"He said"'
```

In [28]:

```
quote
```

Out[28]:

```
'"He said"'
```

In [29]:

```
sentence='''He said "Give me Madhavan's book"'''
```

In [30]:

```
sentence
```

Out[30]:

```
'He said "Give me Madhavan\'s book"'
```

## Use positions, slices etc as for lists

In [31]:

```
s[1:3]
```

Out[31]:

```
'el'
```

In [32]:

```
t[-1]
```

Out[32]:

```
'e'
```

In [34]:

```
l = [0,1,2,3,4]
w = 'abcde'
```

In [35]:

```
l[1], w[1]
```

Out[35]:

```
(1, 'b')
```

In [36]:

```
l[1:2], w[1:2]
```

Out[36]:

```
([1], 'b')
```

In [37]:

```
w[1] == w[1:2]
```

Out[37]:

```
True
```

In [38]:

```
l[1] == l[1:2]
```

Out[38]:

```
False
```

## Strings are immutable

In [39]:

```
s
```

Out[39]:

```
'hello'
```

In [40]:

```
s[3] = 'p'
```

```
---------------------------------------------------------------
-----------------
TypeError                                 Traceback (most
 recent call last)
<ipython-input-40-b756e44260b8> in <module>
----> 1 s[3] = 'p'

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

In [41]:

```
p = s
```

In [42]:

```
p
```

Out[42]:

```
'hello'
```

In [43]:

```
s = 'bye'
```

In [44]:

```
p
```

Out[44]:

```
'hello'
```

In [49]:

```
s = "hello"
s = s[0:3]+ 'p' + s[4:]
```

In [50]:

```
s
```

Out[50]:

```
'helpo'
```

In [54]:

```
s = "   this world   "
```

In [55]:

```
s.strip()  # Removes white space before and after
```

Out[55]:

```
'this world'
```

In [57]:

```
s.lstrip(), s.rstrip()
```

Out[57]:

```
('this world   ', '   this world')
```

## Basic input and output

- Take input from the keyboard
- Print output to the screen

In [59]:

```
x = input()
```

```
The quick brown fox
```

In [60]:

```
x
```

Out[60]:

```
'The quick brown fox'
```

In [61]:

```
x = input()
```

```
67
```

In [62]:

```
x + 45
```

```
---------------------------------------------------------------
-----------------
TypeError                                 Traceback (most
 recent call last)
<ipython-input-62-d316e87d612b> in <module>
----> 1 x + 45

TypeError: can only concatenate str (not "int") to str
```

## Type conversion

- int(s) converts a string s to an int, if it is possible

In [67]:

```
x = input()
y = int(x)+45
```

```
   76
```

In [68]:

```
y
```

Out[68]:

```
121
```

In [76]:

```
x1x2 = input()
```

```
44 77
```

In [85]:

```
l1l2 = x1x2.split()
```

In [86]:

```
y = int(l1l2[0])
z = int(l1l2[1])
y+z
```

Out[86]:

```
121
```

In [84]:

```
y
```

Out[84]:

```
44
```

## Output

- print(x1,x2,...,xn)
- Implicitly each xi is converted to str(xi)

In [89]:

```
print("The original input was",x1x2,"and the split list is",l1l2)
```

```
The original input was 44 77 and the split list is ['44',
'77']
```

In [90]:

```
str(l1l2)
```

Out[90]:

```
"['44', '77']"
```

## Tuples

- (x1,x2,x3)
- Immutable sequence (unlike a list)

In [93]:

```
t = (3,5,7)
```

In [94]:

```
t[1]
```

Out[94]:

```
5
```

In [95]:

```
t[0:2]
```

Out[95]:

```
(3, 5)
```

In [96]:

```
t[0] = 9
```

```
---------------------------------------------------------------
-----------------
TypeError                                 Traceback (most
 recent call last)
<ipython-input-96-9a6f3bfc01f7> in <module>
----> 1 t[0] = 9

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

In [97]:

```
x,y
```

Out[97]:

```
('   76   ', 44)
```

## Mulitple assignment using tuples

In [110]:

```
x,y,z = 0,1,5
```

In [111]:

```
print(x,y,z)
```

```
0 1 5
```

## Exchange the values of two variables

- To swap x and y, normally we need an intermediate temporary value tmp

  tmp = y

  y = x

  x = tmp
- In Python, tuple assignment works!

  (x,y) = (y,x)

In [108]:

```
x,y,z = z,x,y
```

In [109]:

```
x,y,z
```

Out[109]:

```
(1, 5, 0)
```

In [113]:

```
l = [0,1]
s = "hello"
x = 7
l,s,x
```

Out[113]:

```
([0, 1], 'hello', 7)
```

In [114]:

```
l,s,x=x,l,s
```

In [115]:

```
l,s,x
```

Out[115]:

```
(7, [0, 1], 'hello')
```

## More on range and slices

In [117]:

```
list(range(3,10))  # Produce a list from a range
```

Out[117]:

```
[3, 4, 5, 6, 7, 8, 9]
```

In [119]:

```
list(range(11))
```

Out[119]:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [123]:

```
list(range(0,23,2))
```

Out[123]:

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22]
```

In [126]:

```
list(range(23,2,-3)) # Count down
```

Out[126]:

```
[23, 20, 17, 14, 11, 8, 5]
```

In [127]:

```
l = list(range(0,100))
```

In [135]:

```
l[99:10:-5]
```

Out[135]:

```
[99, 94, 89, 84, 79, 74, 69, 64, 59, 54, 49, 44, 39, 34, 2
9, 24, 19, 14]
```

## Recursive functions

- A function can call itself
- "Inductive" definitions
- Standard example is factorial
- n! = n * (n-1) * (n-2) * ... * 1 = n * (n-1)!, need a base case, so 0! = 1 (or 1! = 1)

In [137]:

```
def factrecursive(n):
    if (n == 0):
        return(1)
    else:
        return(n * factrecursive(n-1))   # Call the same function again
```

factrecursive(3) ---- 3 * factrecursive(2) --- suspend and start a new computation

factrecursive(2) ---- 2 * factrecursive(1) --- suspend and start a new computation

factrecursive(1) ---- 1 * factrecursive(0) --- suspend and start a new computation

factrecursive(0) ---- return 1

Resume factrecursive(1) --- 1 * 1` --- return(1)

Resume factrecursive(2) --- 2 * 1 --- return(2)

Resume factrecursive(3) --- 3 * 2 --- return(6)

factrecursive(-1) -- will call -2, -3, -4, ..... --- never terminates

In [138]:

```
def factrecursivefixed(n):
    if (n <= 0):
        return(1)
    else:
        return(n * factrecursive(n-1))   # Call the same function again
```

In [139]:

```
factrecursivefixed(-1)
```

Out[139]:

1

## Recursion on lists

- Base case is empty list

In [141]:

```
def mylength(l):
    if (l == []):
        return(0)
    else:
        return(1 + mylength(l[1:]))
```

In [142]:

```
mylength(list(range(10)))
```

Out[142]:

```
10
```

## Exercise

- Sum of a list of numbers