

Intro to Scientific Programming to:



- Lecture 2

Python standard library

Built-in Functions				
abs ()	delattr ()	hash ()	memoryview ()	set ()
all ()	dict ()	help ()	min ()	setattr ()
any ()	dir ()	hex ()	next ()	slice ()
ascii ()	divmod ()	id ()	object ()	sorted ()
bin ()	enumerate ()	input ()	oct ()	staticmethod ()
bool ()	eval ()	int ()	open ()	str ()
breakpoint ()	exec ()	isinstance ()	ord ()	sum ()
bytearray ()	filter ()	issubclass ()	pow ()	super ()
bytes ()	float ()	iter ()	print ()	tuple ()
callable ()	format ()	len ()	property ()	type ()
chr ()	frozenset ()	list ()	range ()	vars ()
classmethod ()	getattr ()	locals ()	repr ()	zip ()
compile ()	globals ()	map ()	reversed ()	__import__ ()
complex ()	hasattr ()	max ()	round ()	

Example print() and str()

```
# A simple loop that adds 2 to a number  
i = 0  
while i < 10:  
    u = i + 2  
    print('u is ' + str(u))  
    i += 1
```

```
u is 2  
u is 3  
u is 4  
u is 5  
u is 6  
u is 7  
u is 8  
u is 9  
u is 10  
u is 11
```

Note!

Here we format everything to a string before printing it

Python standard library

Built-in Functions				
abs ()	delattr ()	hash ()	memoryview ()	set ()
all ()	dict ()	help ()	min ()	setattr ()
any ()	dir ()	hex ()	next ()	slice ()
ascii ()	divmod ()	id ()	object ()	sorted ()
bin ()	enumerate ()	input ()	oct ()	staticmethod ()
bool ()	eval ()	int ()	open ()	str ()
breakpoint ()	exec ()	isinstance ()	ord ()	sum ()
bytearray ()	filter ()	issubclass ()	pow ()	super ()
bytes ()	float ()	iter ()	print ()	tuple ()
callable ()	format ()	len ()	property ()	type ()
chr ()	frozenset ()	list ()	range ()	vars ()
classmethod ()	getattr ()	locals ()	repr ()	zip ()
compile ()	globals ()	map ()	reversed ()	__import__ ()
complex ()	hasattr ()	max ()	round ()	

In []:

```
width  = 5  
height = 3.6  
snps   = ['rs123', 'rs5487']  
snp     = 'rs2546'  
active = True  
nums    = [2,4,6,8,4,5,2]
```

```
float(width)
```

More on operations

Operation	Result
<code>x + y</code>	sum of x and y
<code>x - y</code>	difference between x and y
<code>x ** y</code>	x to the power y
....
<code>pow(x, y)</code>	x to the power y
<code>float(x)</code>	x converted to float
<code>int(x)</code>	x converted to int!
<code>len(z)</code>	length of z if list
<code>max(z)</code>	maximum in list of z
<code>min(z)</code>	minimum in list of z

In []:

```
x = 4
y = 3
z = [2, 3, 6, 3, 9, 23]
pow(x, y)
```

Comparison operators

Operation	Meaning
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
==	equal
!=	not equal

Can be used on int, float, str, and bool. Outputs a boolean.

In []:

```
x = 5  
y = 3  
  
y > x
```

Logical operators

Operation	Meaning
and	connects two statements, both conditions having to be fulfilled
or	connects two statements, either conditions having to be fulfilled
not	reverses and/or

Membership operators

Operation	Meaning
in	value in object
not in	value not in object

In []:


```
x = 2  
y = 3
```

```
x == 2 and y == 5
```

```
#x = [2, 4, 7, 3, 5, 9]  
#y = ['a', 'b', 'c']
```

```
#2 in x  
#4 in x and 'd' in y
```

```
In [ ]: # A simple loop that adds 2 to a number and checks if the number is even
i      = 0
even = [2,4,6,8,10]
while i < 10:
    u = i + 2
    print('u is '+str(u)+' . Is this number even? '+str(u in even))
    i += 1
```

```
In [ ]: # A simple loop that adds 2 to a number, check if number is even and below 5
i      = 0
even = [2,4,6,8,10]
while i < 10:
    u = i + 2
    print('u is '+str(u)+' . Is this number even and below 5? '+\
          str(u in even and u < 5))
    i += 1
```

Order of precedence

There is an order of precedence for all operators:

Operators	Descriptions
**	exponent
*, /, %	multiplication, division, modulo
+, -	addition, subtraction
<, <=, >=, >	comparison operators
==, !=, in, not in	comparison operators
not	boolean NOT
and	boolean AND
or	boolean OR

Word of caution when using operators

In [8]:

```
x = 5
y = 7
z = 2
x == 5 and y < 7 or z > 1

#x > 6 and (y == 7 or z > 1)

# and binds stronger than or
#x > 4 or y == 6 and z > 3
#x > 4 or (y == 6 and z > 3)
#(x > 4 or y == 6) and z > 3
```

Out[8]: True

In [11]:

```
# BEWARE!
x = 5
y = 8

#xx == 6 or xxx == 6 or x > 2
x > 42 or (y < 7 and xx > 1000)
```

Out[11]: False

Python does short-circuit evaluation of operators

More on sequences (For example strings and lists)

Lists (and strings) are an ORDERED collection of elements where every element can be accessed through an index.

Operators	Descriptions
$x \text{ in } s$	True if an item in s is equal to x
$s + t$	Concatenates s and t
$s * n$	Adds s to itself n times
$s[i]$	i th item of s , origin 0
$s[i:j]$	slice of s from i to $j-1$
$s[i:j:k]$	slice of s from i to $j-1$ with step k

In [7]:

```
l = [2,3,4,5,3,7,5,9]
```

```
n = [9,8]
```

```
s = 'some longrandomstring'
```

Out[7]:

```
9 8 9 8]
```

```
#'o' in s
```

```
#19 in l
```

```
#l + n
```

```
#
```

```
#l[2]
```

```
#s[0:7]
```

```
#s[0:8:2]
```

```
#s[-2]
```

```
#l[0] = 42
```

```
#l
```

```
#s[0] = 'S'
```

Mutable vs Immutable objects

Mutable objects can be altered after creation, while immutable objects can't.

Immutable objects:

- int
- float
- bool
- str
- tuple

Mutable objects:

- list
- set
- dict

Operations on mutable sequences

Operation	Result
<code>s[i] = x</code>	item <i>i</i> of <i>s</i> is replaced by <i>x</i>
<code>s[i:j] = t</code>	slice of <i>s</i> from <i>i</i> to <i>j-1</i> is replaced by the contents of the iterable <i>t</i>
<code>del s[i:j]</code>	removes element <i>i</i> to <i>j-1</i>
<code>s[i:j:k] = t</code>	specified element replaced by <i>t</i>
<code>s.append(x)</code>	appends <i>x</i> to the end of the sequence
<code>s[i:j:k]</code>	slice of <i>s</i> from <i>i</i> to <i>j-1</i> with step <i>k</i>
<code>s[:]</code> or <code>s.copy()</code>	creates a copy of <i>s</i>
<code>s.insert(i, x)</code>	inserts <i>x</i> into <i>s</i> at the index <i>i</i>
<code>s.pop([i])</code>	retrieves the item <i>i</i> from <i>s</i> and also removes it
<code>s.remove(x)</code>	retrieves the first item from <i>s</i> where <code>s[i] == x</code>
<code>s.reverse()</code>	reverses the items of <i>s</i> in place

```
In [8]: s = [0,1,2,3,4,5,6,7,8,9]
        #s.insert(5,10)
        s.reverse()
        s
        #s
```

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

Summary

- The python standard library has many built-in functions regularly used
- Operators are used to carry out computations on different values
- Three types of operators; comparison, logical, and membership
- Order of precedence crucial!
- Mutable object can be changed after creation while immutable objects cannot be changed

Loops in Python

```
In [ ]: fruits = ['apple', 'pear', 'banana', 'orange']

print(fruits[0])
print(fruits[1])
print(fruits[2])
print(fruits[3])
```

```
In [13]: fruits = ['apple', 'pear', 'banana', 'orange']

for fruit in fruits:
    print(fruit)
    print(" was the last fruit")
#     print('end')
print('done')
```

```
apple
  was the last fruit
pear
  was the last fruit
banana
  was the last fruit
orange
  was the last fruit
done
```

Always remember to INDENT your loops!

Different types of loops

For loop

```
In [19]: fruits = ['apple', 'pear', 'banana', 'orange']  
  
for fruit in fruits:  
    print(fruit)
```

apple
pear
banana
orange

While loop

```
In [14]: fruits = ['apple', 'pear', 'banana', 'orange']  
  
i = 0  
while i < len(fruits):  
    print(fruits[i])  
    i = i + 1
```

apple
pear
banana
orange

Different types of loops

For loop

Is a control flow statement that performs a fixed operation over a known amount of steps.

While loop

Is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition.

Which one to use?

For loops better for simple iterations over lists and other iterable objects

While loops are more flexible and can iterate an unspecified number of times

Example of a simple Python script

```
# A simple loop that adds 2 to a number  
i = 0  
while i < 10:  
    u = i + 2  
    print('u is ' + str(u))  
    i += 1
```

```
u is 2  
u is 3  
u is 4  
u is 5  
u is 6  
u is 7  
u is 8  
u is 9  
u is 10  
u is 11
```

Conditional `if/else` statements

Anything that evaluates to a Boolean

```
if condition:  
    print('Condition evaluated to True')  
else:  
    print('Condition evaluated to False')
```

Indentation

```
In [ ]: shopping_list = ['bread', 'egg', 'butter', 'milk']

if len(shopping_list) > 5:
    print('Go shopping!')
else:
    print('Nah! I\'ll do it tomorrow!')
```

```
In [ ]: shopping_list = ['bread', 'egg', 'butter', 'milk']
tired = False

if len(shopping_list) > 5:
    if not tired:
        print('Go shopping!')
    else:
        print('Too tired, I\'ll do it later')
else:
    if not tired:
        print('Better get it over with today anyway')
    else:
        print('Nah! I\'ll do it tomorrow!')
```

This is an example of a nested conditional

Putting everything into a Python script

Any longer pieces of code that have been used and will be re-used SHOULD be saved

Two options:

- Save it as a text file and make it executable
- Save it as a notebook file

Things to remember when working with scripts

- Put `#!/usr/bin/env python` in the beginning of the file
- Make the file executable to run with `./script.py`
- Otherwise run script with `python script.py`

Working on files

```
In [ ]: fruits = ['apple', 'pear', 'banana', 'orange']  
  
for fruit in fruits:  
    print(fruit)
```

```
apple  
pear  
banana  
orange  
fruits.txt (END)
```

```
In [ ]: fh = open('../files/fruits.txt', 'r', encoding = 'utf-8')  
  
for line in fh:  
    print(line)  
  
fh.close()
```

Additional useful methods:

<code>'string'.strip()</code>	Removes whitespace
<code>'string'.split()</code>	Splits on whitespace into list

```
In [ ]: s = '  an example string to split with whitespace in end  '
sw = s.strip()
sw
#l = sw.split()
#l
#l = s.strip().split('\t')
#l
```

```
apple  
pear  
banana  
orange  
fruits.txt (END)
```

```
In [ ]: fh = open('../files/fruits.txt', 'r', encoding = 'utf-8')  
  
for line in fh:  
    print(line.strip())  
  
fh.close()
```

Another example

```
ICA      254
Icecream      65
Coop      25.45
ICA      654.21
Pharmacy      39.90
IKEA      2365
ATM      500
SevenEleven    62.60
ICA      278.50
Åhlens    645.20
bank_statement.txt (END)
```

How much money is spent on ICA?

In []:

```
fh      = open("../files/bank_statement.txt", "r", encoding = "utf-8")

total = 0

for line in fh:
    expenses = line.strip().split() # split line into list
    store    = expenses[0]          # save what store
    price    = float(expenses[1])   # save the price
    if store == 'ICA':               # only count the price if store is ICA
        total = total + price
fh.close()

print('Total amount spent on ICA is: '+str(total))
```


Slightly more complex...

store	year	month	day	sum	
ICA	2018	08	30	254	
Icecream		2018	09	05	65
Coop	2018	09	08	25.45	
ICA	2018	09	22	654.21	
Pharmacy		2018	09	23	39.90
IKEA	2018	09	25	2365	
ATM	2018	09	28	500	
SevenEleven		2018	09	29	62.60
ICA	2018	09	29	278.50	
Åhlens	2018	10	02	645.20	

bank_statement_extended.txt (END)

How much money is spent on ICA in September?

```

In [ ]: fh      = open("../files/bank_statement_extended.txt", "r", encoding = "utf-8")

total = 0

for line in fh:
    if not line.startswith('store'):
        expenses = line.strip().split()
        store     = expenses[0]
        year      = expenses[1]
        month     = expenses[2]
        day       = expenses[3]
        price     = float(expenses[4])
        if store == 'ICA' and month == '09':    # store has to be ICA and month s
eptember
            total = total + price
fh.close()

out = open("../files/bank_statement_results.txt", "w", encoding = "utf-8")    # o
pen a file for writing the results to
out.write('Total amount spent on ICA in september is: '+str(total))
out.close()

```

```

In [ ]: for file in os.scandir("../files/"):
        print(time.ctime(os.stat(file).st_mtime), '\t', file.name)

```

```

Total amount spent on ICA in september is: 932.71
bank_statement_results.txt (END)

```

Summary

- Python has two types of loops, For loops and While loops
- Loops can be used on any iterable types and objects
- If/Else statement are used when deciding actions depending on a condition that evaluates to a boolean
- Several If/Else statements can be nested
- Save code as notebook or text file to be run using python
- The function `open ()` can be used to read in text files
- A text file is iterable, meaning it is possible to loop over the lines

→ **Notebook Day_1_Exercise_4**