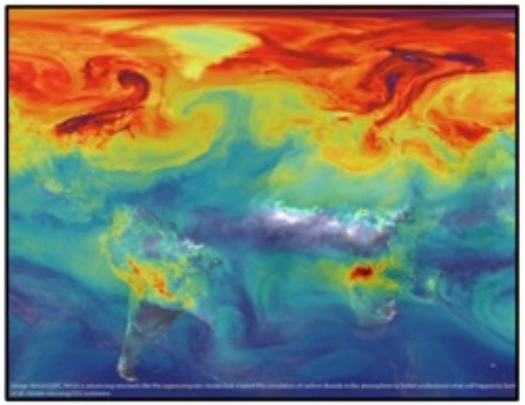





Geo Data Science with Python (GEOS-5984/4984)

Prof. Susanna Werth

Topic: Python Statements – Loops, Iterations and Selections

Today's music is from: Becca

Please keep sending me your song suggestions through Canvas!

Notes/Reminders

- I am out of songs after today!
- SetSnippet_Solutions.py
- Different file types on Jupyter Lab

Filetypes

The following file types are important:

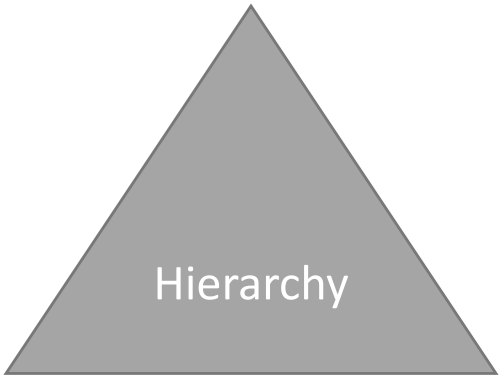
- **.txt** Textfile, e.g., FileTypes.txt
  - will be opened with texteditor automatically
- **.py** Python script file, e.g., HelloWorld.py
  - also textfile, automatically opened with texteditor
  - editor recognizes python code, due to ending .py
  - see the syntax highlighting
  - Application: writing complex programs
- **.ipynb** Jupyter Notebook file, e.g., Lesson01.ipynb
  - JSON source code > open with text editor to see
  - Automatically opened as Notebook
  - Application: Course material, exercises, coding, execute programs, sophisticated documentation of data analysis

Today

- Statements

Python Conceptual Hierarchy

- Python program components
  - Programs are composed of ***modules***
  - Modules contain ***statements***
  - Statements contain ***expressions***
  - Expressions create and process ***objects***



- *Objects* are data elements (e.g. variables, functions, ...)
- *Expression* is a **combination of one or more objects** that the programming language interprets and computes to **produce another object**. They are embedded in statements.
- *Statements* code the larger logic of a program (e.g. assignment, selections, iteration...)
- *Modules* are highest-level organization unit, packages code for reuse

Python Statements

Statements code the larger logic of a program (e.g. assignment, selections, iteration, ...)

- assignments
- `if` Statements
- `while` and `for` loops

Python Statements Handout

Table 10-1. Python statements

Statement	Role	Example
Assignment	Creating references	`a, b = 'good', 'bad'`
Calls and other expressions	Running functions	`log.write("spam, ham")`
`print` calls	Printing objects	`print('The Killer', joke)`
`if/elif/else`	Selecting actions	`if "python" in text:` `  print(text)`
`for/else`	Iteration	`for x in mylist:` `  print(x)`
`while/else`	General loops	`while X > Y:` `  print('hello')`
`pass`	Empty placeholder	`while True:` `  pass`
`break`	Loop exit	`while True:` `  if exittest(): break`
`continue`	Loop continue	`while True:` `  if skiptest(): continue`
`def`	Functions and methods	`def f(a, b, c=1, *d):` `  print(a+b+c+d[0])`
`return`	Functions results	`def f(a, b, c=1, *d):` `  return a+b+c+d[0]`
`yield`	Generator functions	`def gen(n):` `  for i in n: yield i*2`

`global`	Namespaces	`x = 'old'`
		`def function():`
		`  global x, y; x = 'new'`

`nonlocal`	Namespaces (3.X)	`def outer():`
		`  x = 'old'`
		`  def function():`
		`  nonlocal x; x = 'new'`

`import`	Module access	`import sys`
`from`	Attribute access	`from sys import stdin`

`class`	Building objects	`class Subclass(Superclass):`
		`  staticData = []`
		`  def method(self): pass`

`try/except/ finally`	Catching exceptions	`try:`
		`  action()`
		`except:`
		`  print('action error')`

`raise`	Triggering exceptions	`raise EndSearch(location)`
`assert`	Debugging checks	`assert X > Y, 'X too small'`
`with/as`	Context managers (3.X, 2.6+)	`with open('data') as myfile:` `  process(myfile)`

`del`	Deleting references	`del data[k]`
		`del data[i:j]`
		`del obj.attr`
		`del variable`

Lutz (2013), Ch. 10, pp330-331

Combined Assignment Operators

Examples: **a = 5 & b= 2**

Assignment operator	Name	Example	Meaning	Result
=	Simple assignment	a = b	Set a as b	2
+=	Add AND	a += b	a = a +b	7
-=	Subtract AND	a -= b	a= a- b	3
*=	Multiply AND	a *= b	a = a * b	10
/=	Divide AND	a /= b	a = a /b	2
%=	Modulus AND	a % = b	a = a % b	1
**=	Exponent AND	a ** = b	a = a ** b	25

Multiple Assignments & Statements

- The following “multiple assignments” are possible

```
>>> a = b = c = 0
>>> a1, b1, c1 = 1, 1.0, 'c1'
>>> (a2, b2, c2) = 2, 2.0, 'c2'
>>> print a,b,c,a1,b1,c1,a2,b2,c2
0 0 0 1 1.0 c1 2 2.0 c2
```

- Multiple statements on one line

```
>>> a3 = 3; b3 = 3.0 ; c3 = 'c3'
```

Python Statements

Table 10-1. Python statements

Statement	Role	Example
Assignment	Creating references	`a, b = 'good', 'bad'`
Calls and other expressions	Running functions	`log.write("spam, ham")`
`print` calls	Printing objects	`print('The Killer', joke)`
`if/elif/else`	Selecting actions	`if "python" in text:` `  print(text)`
`for/else`	Iteration	`for x in mylist:` `  print(x)`
`while/else`	General loops	`while X > Y:` `  print('hello')`
`pass`	Empty placeholder	`while True:` `  pass`
`break`	Loop exit	`while True:` `  if exittest(): break`
`continue`	Loop continue	`while True:` `  if skiptest(): continue`
`def`	Functions and methods	`def f(a, b, c=1, *d):` `  print(a+b+c+d[0])`
`return`	Functions results	`def f(a, b, c=1, *d):` `  return a+b+c+d[0]`
`yield`	Generator functions	`def gen(n):` `  for i in n: yield i*2`

`global`	Namespaces	`x = 'old'`
		`def function():`
		`  global x, y; x = 'new'`

`nonlocal`	Namespaces (3.X)	`def outer():`
		`  x = 'old'`
		`  def function():`
		`  nonlocal x; x = 'new'`

`import`	Module access	`import sys`
`from`	Attribute access	`from sys import stdin`

`class`	Building objects	`class Subclass(Superclass):`
		`  staticData = []`
		`  def method(self): pass`

`try/except/ finally`	Catching exceptions	`try:`
		`  action()`
		`except:`
		`  print('action error')`

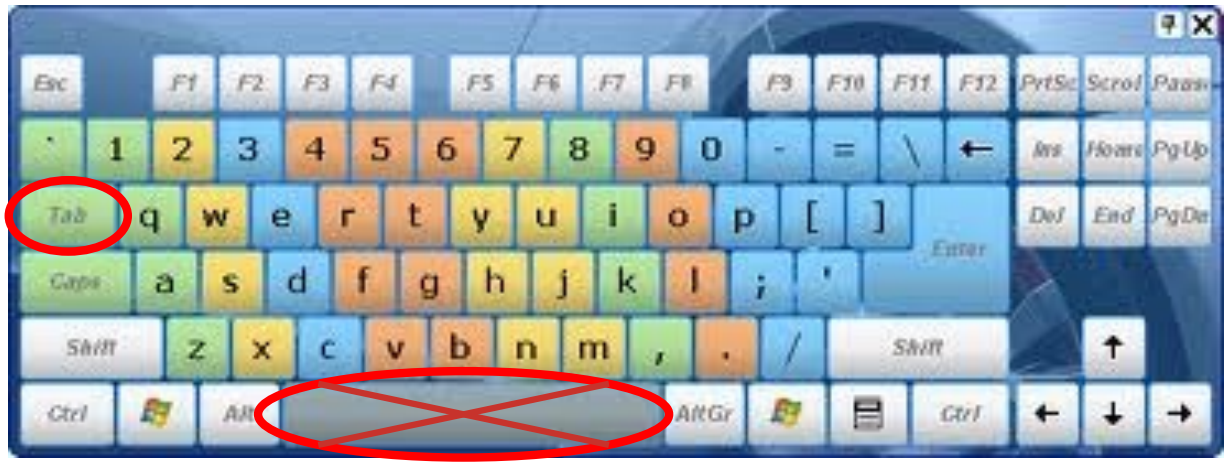
`raise`	Triggering exceptions	`raise EndSearch(location)`
`assert`	Debugging checks	`assert X > Y, 'X too small'`
`with/as`	Context managers (3.X, 2.6+)	`with open('data') as myfile:` `  process(myfile)`

`del`	Deleting references	`del data[k]`
		`del data[i:j]`
		`del obj.attr`
		`del variable`

Lutz (2013), Ch. 10, pp330-331

Indentation: Space vs Tabs

- Indentation is very important in Python for **grouping statements** into different levels
- "4 spaces" or **1 TAB** are usually used to indent different level of statements
- You *can not* safely mix space and tabs in Python, and avoid using spaces if possible



Variable Names

Case Sensitivity

Python is case sensitive on all names, e.g., classes, objects, variables etc.

Keywords/Reserved Words

Keywords can not be used as the names of variables, classes, and objects etc.

A Syntax Error will occur !

Table 11-3. Python 3.0 reserved words

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

General `if` Statements

```
if test1:
   statements
elif test2:
   statements
elif test3:
   statements
...
else:
   statements
```

- **Syntax: colon, indent**
- test conditions evaluate to True or False
- `elif` introduces second test
- `else` introduces alternative action
- `elif` and `else` components are optional
- Sequential Hierarchy!

Tutorial

Selection 1: What will be printed out, A, B, C or D?

Find code in ***StatementSnippet.py***

```
yesterday = 14
today = 13
tomorrow = 13

if yesterday < today:
    print('A: yesterday was colder than today')
elif today != tomorrow:
    print('B: yesterday was not the same temperature as today')
elif yesterday > tomorrow:
    print('C: yesterday was warmer than today')
elif today == today:
    print('D: yesterday and today had equal temperatures')
```

Tutorial

Selection 2: Boolean tests and (multiple) selections

```
weather = 'Rain' # 'Sun', 'Clouds'
wind = 'Windy'   # 'notWindy'
```

1. Write a selection that tells you to just stay home, if it is `Rain` and `Windy` => *multiple conditions*
2. Add `elif` option for only `Rain`: wearing a raincoat, otherwise no raincoat needed. => *multiple selections*

Compound Statements

```
header expression:  #(e.g., for/while/if)

   # Nested statement block

   if booleanTest == true:
        print “this”
   else:
        print “that”
```

Special Case: Ternary `if` Statements

```
if test1:
    x = A
else:
    x = B
```

```
x = A if test1 else B
```

Tutorial

Selection 3: Ternary if

Rewrite the temperature selection into a ternary if statement. Set temperature to any value.

```
if temperature > 25:
    print('it is hot')
```

Python Statements

Table 10-1. Python statements

Statement	Role	Example
Assignment	Creating references	`a, b = 'good', 'bad'`
Calls and other expressions	Running functions	`log.write("spam, ham")`
`print` calls	Printing objects	`print('The Killer', joke)`
`if/elif/else`	Selecting actions	`if "python" in text:` `  print(text)`
`for/else`	Iteration	`for x in mylist:` `  print(x)`
`while/else`	General loops	`while X > Y:` `  print('hello')`
`pass`	Empty placeholder	`while True:` `  pass`
`break`	Loop exit	`while True:` `  if exittest(): break`
`continue`	Loop continue	`while True:` `  if skiptest(): continue`
`def`	Functions and methods	`def f(a, b, c=1, *d):` `  print(a+b+c+d[0])`
`return`	Functions results	`def f(a, b, c=1, *d):` `  return a+b+c+d[0]`
`yield`	Generator functions	`def gen(n):` `  for i in n: yield i*2`

`global`	Namespaces	`x = 'old'`
		`def function():`
		`  global x, y; x = 'new'`

`nonlocal`	Namespaces (3.X)	`def outer():`
		`  x = 'old'`
		`  def function():`
		`  nonlocal x; x = 'new'`

`import`	Module access	`import sys`
`from`	Attribute access	`from sys import stdin`

`class`	Building objects	`class Subclass(Superclass):`
		`  staticData = []`
		`  def method(self): pass`

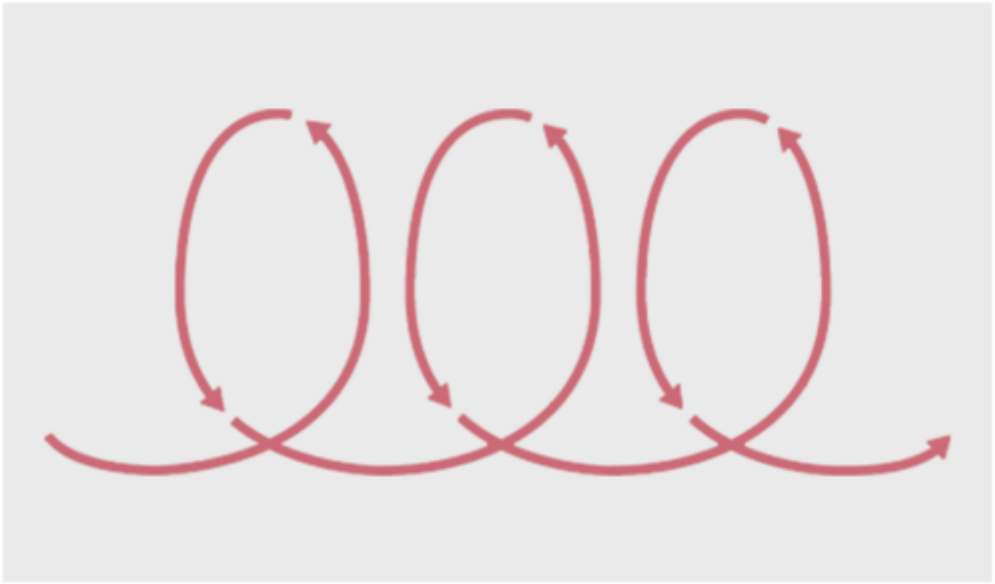
`try/except/ finally`	Catching exceptions	`try:`
		`  action()`
		`except:`
		`  print('action error')`

`raise`	Triggering exceptions	`raise EndSearch(location)`
`assert`	Debugging checks	`assert X > Y, 'X too small'`
`with/as`	Context managers (3.X, 2.6+)	`with open('data') as myfile:` `  process(myfile)`

`del`	Deleting references	`del data[k]`
		`del data[i:j]`
		`del obj.attr`
		`del variable`

Lutz (2013), Ch. 10, pp330-331

Iteration / Loop



Iteration is the:

- Repetition of a process
- Each repetition of the process is a single iteration, and the outcome of each iteration is then the starting point of the next iteration.
- Example: applying the same set of expressions and statements to several variables, e.g. several items of a list, or other sequential object types.

Example: Computing the Sample Mean

Consider the problem of computing the sample mean using a finite sample from a population.

Let $X = [x_1, x_2, x_3, \ldots, x_n]$ be a sample of size n from a population.

Example sample: $X = [123, 87, 96, 24, 104, 16]$

Sample mean: $$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

General `for` Statements

```
for target in object:
    statements
else:
    statements
```

- Iterates through the items in the `object` assigning each value to `target`.
- `else + statements` (optional) – execute if the for didn't end with a break

- Examples for `object`:
  - Iterables: list, tuple
  - Dictionary keys, set...
  - a range: range(startval,enval+1)

- Similar to list comprehension

Tutorial

Iteration 1: Calculate the Sample Mean!

$X = [123, 87, 96, 24, 104, 16]$

Write an iteration to calculate the mean of the sample.

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Advanced: use the random.random() function to generate a list of 10 random numbers in an iteration. Then calculate the mean of the random list.

General `while` Statements

```
while test:
    statements
```

- Indefinite loops (unknown number of iterations)
- test conditions evaluate to True or False.
- *`statements`* execute repeatedly until test fails (evaluates to False)

General `while` Statements

```
while test:
    statements
else:
    statements
```

- `else + statements` (*optional*) – execute if the loop is not interrupted

Tutorial

Iteration 2:
Check the password!

1. Define a password string.
2. Now, let's create a `while` loop, which asks for the user to input a password using the function 'input()'.
3. While going through this loop, there are two possible outcomes:
   - If the password is correct, the while loop will exit.
   - If the password is not correct, the while loop will continue to execute.

Python Statements

Table 10-1. Python statements

Statement	Role	Example
Assignment	Creating references	`a, b = 'good', 'bad'`
Calls and other expressions	Running functions	`log.write("spam, ham")`
`print` calls	Printing objects	`print('The Killer', joke)`
`if/elif/else`	Selecting actions	`if "python" in text:` `  print(text)`
`for/else`	Iteration	`for x in mylist:` `  print(x)`
`while/else`	General loops	`while X > Y:` `  print('hello')`
`pass`	Empty placeholder	`while True:` `  pass`
`break`	Loop exit	`while True:` `  if exittest(): break`
`continue`	Loop continue	`while True:` `  if skiptest(): continue`
`def`	Functions and methods	`def f(a, b, c=1, *d):` `  print(a+b+c+d[0])`
`return`	Functions results	`def f(a, b, c=1, *d):` `  return a+b+c+d[0]`
`yield`	Generator functions	`def gen(n):` `  for i in n: yield i*2`

`global`	Namespaces	`x = 'old'`
		`def function():`
		`  global x, y; x = 'new'`

`nonlocal`	Namespaces (3.X)	`def outer():`
		`  x = 'old'`
		`  def function():`
		`  nonlocal x; x = 'new'`

`import`	Module access	`import sys`
`from`	Attribute access	`from sys import stdin`

`class`	Building objects	`class Subclass(Superclass):`
		`  staticData = []`
		`  def method(self): pass`

`try/except/ finally`	Catching exceptions	`try:`
		`  action()`
		`except:`
		`  print('action error')`

`raise`	Triggering exceptions	`raise EndSearch(location)`
`assert`	Debugging checks	`assert X > Y, 'X too small'`
`with/as`	Context managers (3.X, 2.6+)	`with open('data') as myfile:` `  process(myfile)`

`del`	Deleting references	`del data[k]`
		`del data[i:j]`
		`del obj.attr`
		`del variable`

Lutz (2013), Ch. 10, pp330-331

Loop Breakers

- `Break`
  - Terminates loop
  - For and while loops
- `Continue`
  - Returns back to start of loop statement
  - For and while loops
- `Pass`
  - Used when syntactically a statement is required, but don't want to anything to happen

```
for i in range (10):
    if i==3:
        break
    print(i)
# ends iteration at i == 3
```

```
for i in range (10):
    if i==3:
        continue
    print(i)
# skips printing at i == 3
```

```
for i in range (10):
    if i==3:
        pass
    print(i)
# does print at i == 3
```

Tutorial

Check the password!

4. + ELSE: Add final feedback to the user by coding an else statement for the while loop, when it is exited.
5. + COMOUND STATEMENTS (optional):
    - Add a counter and interrupt the password input after three false trials.
    - Return final feedback whether password is correct or not.
6. + LOOP BREAKERS

Practice

- Mandatory:
  - Revise slides and notebook section B2.4 about loop breakers. You can also look at the syntax provided in the handout for statements!
  - Complete the indefinite loop for password check, including else and loop breakers.

- Optional: Revise L07 notebook on Expressions and Statements