

# Empezando con Python/Pitón

Andrew Reeve  
School of Earth and Climate Sciences  
University of Maine

# ¿Qué es Python?, ¿Qué se puede hacerlo?

Idioma informático interpretado, ¡gratificación instantánea!

## Bibliotecas incorporadas, y bibliotecas externas

- SIG (Shapely, cartopy)
- Rutinas científicas (Scipy)
- Algoritmos de Aprendizaje Automático, Estadísticos (Scikit-Learn, Statsmodels)
- IGU's (Tk, Qt)
- Juegos (Pygame, Pyglet)
- Raspado de Pantalla (Beautiful Soup, scrapy)
- Matemáticas Lujosas (Numpy, Sympy)

## Estilos de Programación

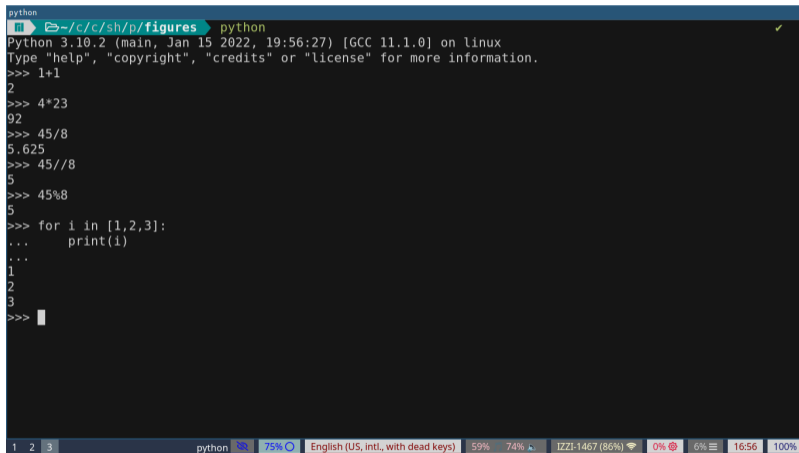
- Procesal (Procedural)
- Orientado a Objetos (Object Oriented)
- Funcional (Functional)
- Extensible

## Herramientas para el Codificar

- Shell Interactivo de Python
- Shell de Ipython
- Cuadernos de Jupyter
- Editores de texto, Entorno de desarrollo integrado (IDE's)

# Usando el shell de Python

- Escribe 'python' en un terminal o haga clic en el icono:



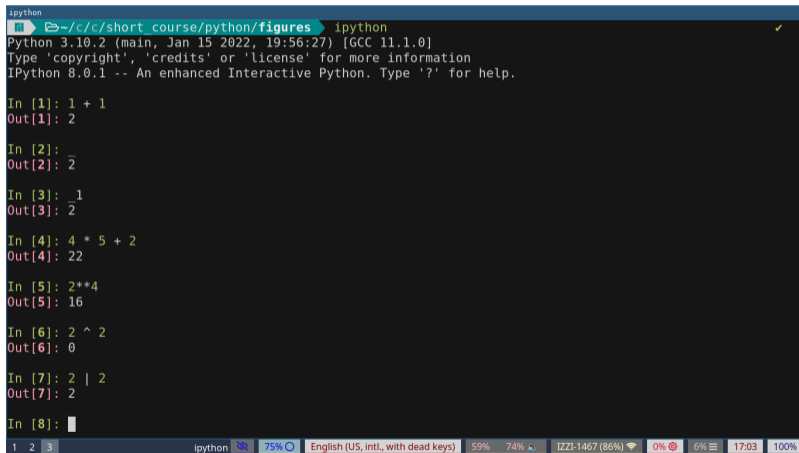
The screenshot shows a terminal window with a dark background. At the top, the title bar reads 'python'. The terminal content is as follows:

```
python
~/c/c/sh/p/figures python
Python 3.10.2 (main, Jan 15 2022, 19:56:27) [GCC 11.1.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 1+1
2
>>> 4*23
92
>>> 45/8
5.625
>>> 45//8
5
>>> 45%8
5
>>> for i in [1,2,3]:
...     print(i)
...
1
2
3
>>> █
```

The terminal window has a status bar at the bottom with the following information: '1 2 3', 'python', '75% O', 'English (US, Intl., with dead keys)', '59%', '74%', 'IZZI-1467 (86%)', '0%', '6%', '16:56', and '100%'.

# Usando el shell de Ipython

- Escribe 'ipython' en un terminal o haga clic en el icono:



```
ipython
~/c/c/short_course/python/figures ipython
Python 3.10.2 (main, Jan 15 2022, 19:56:27) [GCC 11.1.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.0.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: 1 + 1
Out[1]: 2

In [2]: _
Out[2]: 2

In [3]: _1
Out[3]: 2

In [4]: 4 * 5 + 2
Out[4]: 22

In [5]: 2**4
Out[5]: 16

In [6]: 2 ^ 2
Out[6]: 0

In [7]: 2 | 2
Out[7]: 2

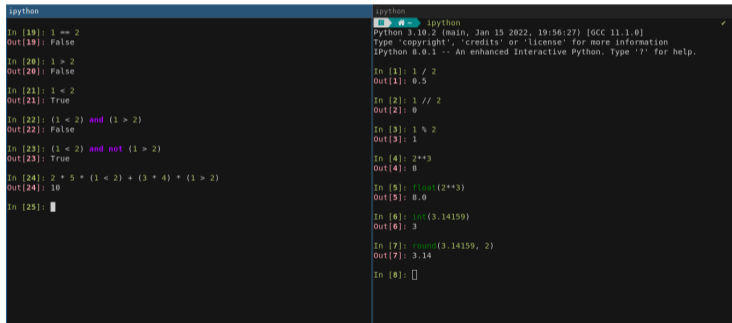
In [8]: █
```

The screenshot shows a terminal window with a dark background. At the top, the terminal title is 'ipython'. The prompt shows the current directory as '~/c/c/short\_course/python/figures' and the command 'ipython' has been executed. The output shows the Python version (3.10.2), IPython version (8.0.1), and a list of arithmetic operations with their results. The operations are: 1 + 1 = 2, 2 (from the previous result), 2 (from the previous result), 4 \* 5 + 2 = 22, 2\*\*4 = 16, 2 ^ 2 = 0, and 2 | 2 = 2. The terminal also shows a status bar at the bottom with various system icons and settings.



# Operaciones de Matemáticas Básicas

- matemáticas [adición(+), división(/), *división de piso* (/), multiplicación(\*), exponente(\*\*), módulo(%)]
- booleanos: Sí/No, verdadero/falso; 1/0 [and, or, not, in, is, ==, !=, <, >]; puede usar en ecuaciones.
- otras operaciones básicas: float, int, abs, round



```
ipython
In [19]: 1 == 2
Out[19]: False

In [20]: 1 > 2
Out[20]: False

In [21]: 1 < 2
Out[21]: True

In [22]: (1 < 2) and (1 > 2)
Out[22]: False

In [23]: (1 < 2) and not (1 > 2)
Out[23]: True

In [24]: 2 * 5 + (1 < 2) + (3 * 4) * (1 > 2)
Out[24]: 10

In [25]: █

ipython
Python 3.10.2 (main, Jan 15 2022, 19:56:27) [GCC 11.1.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.0.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: 1 / 2
Out[1]: 0.5

In [2]: 1 // 2
Out[2]: 0

In [3]: 1 % 2
Out[3]: 1

In [4]: 2**3
Out[4]: 8

In [5]: float(2**3)
Out[5]: 8.0

In [6]: int(3.14159)
Out[6]: 3

In [7]: round(3.14159, 2)
Out[7]: 3.14

In [8]: █
```

# Tipos de Datos

## Floats and Ints

```
1 # entero (an integer)
2 i=10
3 # número de punto
  flotante (a float)
4 x=1.99
5 # cadena de
  caracteres, texto
  (string)
6 txt='hello nasty
  world'
print(f'{i} es entero
')
```

Output  
10 es entero

Andrew S Reeve

## Cadenas de caracteres, texto (Strings)

```
1 # a sting
2 txt='\hello \t nasty
  \n world'
3 # '\ ' indica un
  carácter especial
  (tab, newline)
4 print(txt)
```

Output  
\hello nasty  
world

Data Types, Variables, and Basic Calculations

## Datos Booleano

```
1 # a boolean
2 B1=True
3 B2=False
4 print('test1:', B1 ==
  B2)
5 print('test2:', B1 !=
  B2)
6 # math with boolean
  values!
7 print(B1 * B1 + 2)
8 # logic tests on
  numbers
9 print('is 1 bigger
  than 2?:', 1 > 2)
```

Output  
test1: True  
test2: True

7/32

## Determinando el Tipo de Dato

Usa la manda 'type' para determinar el tipo del dato almacenado en un variable.  
¿Porqué es importante el tipo de dato?

- cambia como los operaciones funciona.
- métodos o funciones diferentes están asignados a tipos de datos diferentes.
  - usa una sintaxis de punto para acceso diferentes funciones

## Ejemplos:

```
1 B=True #boolean
2 I=1 #integer
3 S='El Mundo Malo'
4 print(type(B),type(I),type(S))
5 print(I+I)
6 print(I+.2)#upcasting
7 print(S+S)
8 W=S.swapcase()#a method for strings
9 print(W)
```

Output

```
<class 'bool'> <class 'int'> <class 'str'>
2
1.2
El Mundo MaloEl Mundo Malo
eL mUNDO mALO
```



# Contenedor/Envase (Tipos de Datos Compuestos)

## Las Estructuras de Datos

- Estructuras de Datos que almacenan muchos artículos
- cada estructura tiene sus propios métodos y atributos
- cadenas de caricaturas funcionan así, un envase de caracteres
  - almacena solamente caracteres (no números)
  - inmutable
- Algunos Envases permite la selección:  
[start:end:stride]
- indice empieza con 0 (item 0 is first item)

```
# define a string , get
length
1 a = 'hellohola'
2 print(len(a))
3
4 # get character , count
number of 'h' vals
5 print(a[0], a.count('h'))
6 # slice string
7 print(a[5:9], a[5:], a[:5])
8 # all caps
9 print(a.upper())
```

Output

```
9
h 2
hola hola hello
HELLOHOLA
```

## List Properties

- paréntesis cuadrado o list functiona
- mezcla de tipos de datos (strings, floats, lists...)
- mutable (puede modificar)
- muchos métodos para modificar

```
# make a list and extract
elements
2 lst = [1,2, 'hello', 'adios'
        ,[3.,4.]]
print(lst[0], lst[2][1:],
      list((lst[0:2])))
4 # add to a list
lst.append('¿cómo?')
6 lst.append('what?')
print(lst)
8 # sort and reverse
lst.reverse()
10 print(lst)
```

Output

```
1 ello [1, 2]
[1, 2, 'hello', 'adios', [3.0, 4.0], '¿cómo?', 'what?']
['what?', '¿cómo?', [3.0, 4.0], 'adios', 'hello', 2, 1]
```

## Propiedades de tuplas

- paréntesis o tuple function
- inmutable
- mezcla de tipos
- no hay muchos métodos

```
# make a list and extract  
elements
```

```
2 tpl = (1, 2, 'tres')  
3 print(tpl[2])  
4 print(tpl.count(2))
```

Output

```
tres  
1
```

## Propiedades de Diccionarios

- paréntesis rizado o dict function
- arreglos asociados, pares de llave:valor
- mutable
- muchos métodos

```
albite = { 'formula' : 'NaAlSi4O8' , '
          hardness' : '6.5' , 'cleavage' : '2' }
2 quartz = { 'formula' : 'SiO2' , 'hardness
            ' : '7' , 'cleavage' : 'None' }
halite = { 'formula' : 'NaCl' , 'hardness
          ' : '2.5' , 'cleavage' : '3' }
4 minerals = {}
minerals [ 'quartz' ] = quartz
6 minerals . update ( { 'albite' : albite } )
print ( '1' , minerals )
8 print ( minerals . keys ( ) )
print ( minerals . values ( ) )
```

Output

```
1 {'quartz': {'formula': 'SiO2', 'hardness': '7', 'cleavage': 'None'}, 'albite': {'formu
dict_keys(['quartz', 'albite'])
dict_values(['formula': 'SiO2', 'hardness': '7', 'cleavage': 'None'], {'formula': 'NaAl
```

# Ejemplos

```
1 albite = { 'formula': 'NaAlSi4O8', 'hardness': '6.5', 'cleavage': '2' }
2 quartz = { 'formula': 'SiO2', 'hardness': '7', 'cleavage': 'None' }
3 halite = { 'formula': 'NaCl', 'hardness': '2.5', 'cleavage': '3' }
4 minerals = { 'albite': albite, 'quartz': quartz }
5 minerals.setdefault('quartz', halite)
6 print('2', minerals) #.setdefault doesn't update quartz
7 minerals.setdefault('halite', halite)
8 print('3', minerals) # updated now b/c 'halite' not in dict
```

Output

```
2 {'albite': {'formula': 'NaAlSi4O8', 'hardness': '6.5', 'cleavage': '2'}, 'quartz': {'formula': 'SiO2', 'hardness': '7', 'cleavage': 'None'}}
3 {'albite': {'formula': 'NaAlSi4O8', 'hardness': '6.5', 'cleavage': '2'}, 'quartz': {'formula': 'SiO2', 'hardness': '7', 'cleavage': 'None'}, 'halite': {'formula': 'NaCl', 'hardness': '2.5', 'cleavage': '3'}}
```

## Set Properties

- paréntesis rizadoa
- sin orden
- solo un valor en cada conjuntos (no permite duplicación)
- mutable

```
a = {1, 2, 3, 3}
2 b = set((3, 4, 5))
   print(a, b)
4   print(a.union(b))
   print(a.intersection(b))
6   print(a.difference(b))
   print(b.difference(a))
```

Output

```
{1, 2, 3} {3, 4, 5}
{1, 2, 3, 4, 5}
{3}
{1, 2}
{4, 5}
```

## Propósito y las Reglas para los Variables

- los variables almacenen los datos; usa un nombre descriptivo
- es como pone información en una caja, el variable es la caja
  - en una ubicación para almacenar información
  - el variable no es la información, apunta a la información
- asignado con '='
- el primero carácter debe se una letra o un carácter subrayado, usa un nombre razonable
- defina ecuación en términos de otros variables
- puede crear 'expresiones circulares', usa valor viejo para define valor nuevo (e.g.  $i=i+2$ )

## Ejemplos

```
# an integer  
2 ten=10  
class_number=420  
4 class_dept='ers'  
class_value=0.25 #  
dollars
```

El texto después de '#' es una comenta. Todos los valores tienen un tipo de dato que se determinen dinámicamente.

# Programming Logic and Repetition; Functions; Importing Modules

Andrew Reeve  
School of Earth and Climate Sciences  
University of Maine



## Declaraciones de if, elif, else

- `if` Declaración Booleana :
- la sangría define a donde termina
- bloque de declaraciones ejecutan si está `True`
- condiciones con bloques de `elif`
- solo el primero `True` bloque ejecuta
- `else` bloque ejecuta si no Declaración Booleana está verdad

```
i=3
2 if i > 0:
    print('i is positive')
4 elif i < 0:
    print('i is negative')
6 elif i < 10:
    print('i is greater than 10')
8 else:
    print('i is zero')
```

# Introducción a Funciones

- Funciones son pequeños programas se llama para ejecutar una tarea (o tareas)
- las has usado cuando llama `print('some text')`, `int(5.5)`, y otras declaraciones
  - `print` es una función
  - `'some_text'` es el argumento
- creado con la palabra `def`
  - `def func_nombre(arg1,arg2,arg3):`  
código que quiere ejecutar `return` que quiere devolver
  - una paquete de código reutilizarse (DRY)
  - llamado con `func_nombre`
  - salida al programa principal con palabra `return`

```
def is_odd(number):  
    2     if number%2==1:  
           return True  
    4     else:  
           return False  
    6  
number = 2  
    8     print(is_odd(number))
```

Output

False

## Usando las Funciones

- Módulos son guiones de pitón que contiene funciones
- puede usa después de importar con la palabra `import`
- puede obtener un lista de los módulos instalados con `import` seguido con la tecla 'tab' (tab completion)

Si el ultimo guion estaba ahorrado con `is_odd.py`

---

---

```
import sys
2 sys.path.append('../examples')
import is_odd
4
num_5=is_odd.is_odd(5)
6 print(num_5)
```

---

---

# Argumentos de Función

## Argumentos de Palabra Clave

- asigna el argumento con palabra clave
- puede definir argumentos predeterminados
- despues de argumentos posicional

## Variable Arguments

- pasa múltiples argumentos con un valor
- el numero de argumentos pueden cambiar
- un asterisco, pasa una tupla
- dos asteriscos, pasa una diccionario
- el orden importa: 1. positional, 2. \* var.args, 3. palabra clave, 4. \*\* var. args

```
def example1(*data):  
    print(data)  
example1(1,2,3)  
example1(1,2,3,4)  
  
def example2(a,*data):  
    print(a)  
    print(data)  
example2(1,2,3)  
  
def example3(a,*data,b='2nd'):  
    print(a,b,data)  
example3(1,2,3)
```

# Más Ejemplos

```
def example3(**data):
    print(data)
    print('The grade for bill is {}'.
          .format(data['bill']))
    example3(bill='A', fred='C')

# order here matters
def example4(a,b='hello',**data):
    print(a,b)
    print(data)
    example4(1,z=2,y=3)
    example4(1,b='goodbye',z=2,y=3)
```

```
#!/usr/bin/env python3
2 import math as mt

4 def apparent2truedip(A,B,radians=0):
    ...

6     This docsting is placed at start of function and describe what the functions
        does, arguments, returned items, etc.
        Calculates true dip from apparent dip:

8     Input
        -----
10     A [float] apparent dip
12     B [float] angle between apparent and true dip directions
        radians [boolean] default is False

14     Output
        -----
16     C=true dip
        ...

18     if radians==1:
20         C=mt.atan(mt.tan(A)/mt.cos(B))
        else:
22         A=mt.radians(A)
24         B=mt.radians(B)
26         C=mt.atan(mt.tan(A)/mt.cos(B))
28         C=mt.degrees(C)
        return C

# function has been defined and can now be used
28 if __name__=='__main__':
30     A=45.
32     B=20.
    C=apparent2truedip(A,B,radians=0)
    print(C)
```

# Funciones dentro de Funciones

## Funciones y Recursivo

- funciones pueden llamar otras funciones
- pueden llamarse a sí mismas, un patrón recursivo
  - se llama a sí misma que se llama a sí misma...

## Ejemplo

```
def afunc(n):  
2   if n==0:  
    return 1  
4   else:  
    return (1/n)*afunc(n-1)  
6  
8   if __name__ == '__main__':  
    print(afunc(5))
```

## func en func

```
import operator  
2 def calc_something(op):  
   ...  
4     return a function that does a basic operation  
  
6     can change meaning or returned function based in  
   input  
   ...  
8     # map the basic functions to strings  
10    get_op={'+' : operator.add,  
            '-' : operator.sub}  
12    return get_op[op]  
  
14 if __name__ == '__main__':  
    my_op=calc_something('+')  
16    print(my_op(1,2))
```

# Bucles de For/Else

## Bucles de For

- iteración sobre cada ítem en un envase
- iteración sobre los generadores
- variable asignado en orden consecutivo a los valores en un iterable
- final `else` está ejecutado si el bucle termina

```
# item is a variable name, iterable after 'in',  
# ends w/ colon  
2 for item in ['file1', 'file2', 'file3']:  
# do something to iterable  
4     do_stuff = item[-1]*int(item[-1])  
     print(do_stuff)  
6 else:  
     print("will always print, no break in for  
     loop!")
```

Output

```
1  
22  
333  
will always print, no break in for loop!
```

## Range, Zip y Enumerate

- `range` crea un generador
  - usado comúnmente used
  - start, stop, stride (separado por comas)
  - números creado sobre la marcha
- `enumerate` asigna un numero con el iterable
  - en orden consecutivo devuelve (numero, ítem)
- `zip` combinar iterables de la misma longitud
  - 'zips' together, cremalleras juntas iterables diferentes
  - todos los ítems de los iterables devuelven en orden (1°, 2°, 3°...)

```
letters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
2
letter_list = []
4 number_list = []
for i, ltr in enumerate(letters):
6     print(i, ltr, end=' ')
    number_list.append(i)
8     letter_list.append(ltr)
    if i > 8:
10        print('')
        break
12
for i, ltr in zip(number_list,
14        letter_list):
    # prevent newline with end argument
    print(f'{i*2},{ltr*2}', end=" ")
```



## Break, Continue, y Pass

Puede modificar la estructura de un bucle con las mandas:

- `break`: romper un bucle
- `continue`: va al próximo ciclo de bucle
- `pass`: no pasa nada

# Ejemplos de Bucles de For

## Example (ejemplos else-break)

```
# when does x**n=7?
2 x=10 # a guess
n=3 # change this to make unstable
4
for i in range(20):
6     slope=((x+.005)**n - (x-.005)**n)/0.01
    y=x**n-7 # this is measure of error
8     x=x-y/slope
    if y>1e6:
10         print('answer not found')
        break
12 else:
    print(f'if x={x}, x**{n}=7')
```

## Example (ejemplos break-continue-pass)

```
# find numbers less than 'n'
2 # that are not divisible by 2, 3 or 5
n=20
4 for i in range(n):
    if i>100:
6         print(f'ended loop with break at {i}')
        break
8     if i%2==0 or i%3==0 or i%5==0:
        continue
10 print(i)
12 else:
    print(f'ended loop at {i}')
```

## Crea un bucle de While

- while declaración booleana :
- ciclo hasta la condición en while declaración está False
- en ipython, Ctrl-c termina ciclo infinito
- final else cláusula está ejecutada si no esta roto

```
import random # access random numbers
2 time=0
  population = 1e6
4 infected = 10
  infect_rate=.01
6 recov_rate=0.05
  while time < 100:
8     time = time + 1
      new_cases=0
10    for infect in range(infected):
      if random.uniform(0,1)<infect_rate:
12         new_cases = new_cases + 1
          new_recov=0
14    for infect in range(infected):
      if random.uniform(0,1)<recov_rate:
16         new_recov = new_recov + 1
          infected = infected + new_cases -new_recov
18    if infected <=0: # disease eradicated
      break
20    elif infected >1e5: # disease out of control
      break
22    print(time)
  else:
24     print('did not break loop')
      print(f'{infected} cases after {time} steps')
```

## Comprensión de Lista

- una manera compacto para crear una lista (o dict, tupla, generador)
- un bucle adentro una lista (o dict, tupla...)

```
import random
2 # make list of 100 random numbers
  random_uniform = [random.uniform(0,1) for i in range(100)]
4 # including if statement
  random_index = [i for i in range(100) if random_uniform(0,1) <.1]
6 # getting the random numbers selected
  select_random = [round(i,3) for i in [random.uniform(0,1) for i in range(100)]
                    if i <.1]
8 print(random_index)
  print(select_random)
10 # can also do with other containers
  d1={a:b for a,b in [[1,2],[ 'one', 'two' ]]}
12 d2={a:b for a,b in zip([1,2],[ 'one', 'two' ])}
  print(d1)
14 print(d2)
```

# Digresión: formato de cadena

- varias maneras para empotrar los variables en una cadena
  - - % formato (más viejo)
  - método de 'format'
  - cadenas de 'f' (más reciente)
    - 'f' antes de comillas alrededor de la cadena

```
Output
a=1          b=2.0          c=3000.0
a=100.00%    b=0000002, c=3000.0
```

```
a='1'
2 b=2.000
c=3.e3
4 # f-strings
print(f' a={a}\t b={b}\t c={c}')
6 print(f' a={int(a):.2%}\t b={b:07n}, c={c}\n')
# with format command
8 print(' a={2:.1e}, b={0}, c={1}'.format(a,b,c))
# % formatting
10 print(' a=%s\n b=%s\n c=%s\n'%(a,b,c))
```

```
Output
a=1          b=2.0          c=3000.0
a=100.00%    b=0000002, c=3000.0

a=3.0e+03, b=1, c=2.0
a=1
b=2.0
c=3000.0
```

## Digression 2: input

### input command

- `input`: prompts and captures keyboard input
- `a = input(prompt)`
  - `a` assigned input
  - `prompt` is displayed
  - always is string

---

---

```
new_string = input('please enter something')
```

---

---

# Making a one question survey

## Approach & Description

- Docstring
  - a multi-line string
  - start & end with triple quotes
- Request input from command line
  - make sure it is sensible input
- Test how excited you are based on input
  - three conditions
  - make a string based on input
  - print something based on input

```
'''  
2 code example 3: using if statements,  
   Remember only one block is executed  
4 '''  
  
6 response = input('''' How excited are you?  
   Enter an integer number between -10 and 10.  
8 10 is super excited, -10 is totally uninspired.'''')  
  
10 response=int(response)  
   print(response)  
12  
13 if response > 0:  
14     excitement='s' + response * 'o'  
   print(f"I\'m {excitement} excited")  
16 elif response == 0:  
   print("I\'m ambivalent.")  
18 else:  
   excitement=abs(response) * ' really '  
20   print(f"I\'m {excitement} bored")
```

# Declaraciones Booleanas

## Booleanas

- statements that return True or False
- numbers, 0 or 0.0 are False
- can use and, or, and not to chain together and modify boolean statement

```
if True and False:
    print('and yes')
elif True or False:
    print('or yes')
else:
    print('else')
```

Output

or yes