Università
degli Studi
della Campania
*Luigi Vanvitelli*

# Iterables: List, Tuple, Set, Dictionaries

Ing. Lelio Campanile

# Introducing lists

- A list is a collection of items, stored in a variable

- Items should be related

- Items could be of diffrent types (objects)

```
colors = ['red', 'green', 'blue']
print(colors)
['red', 'green', 'blue']
```

# Naming a list

- since lists are collection of objects, it is a good practice to give them a plural name
- for a list of color, call the list 'colors'
- for a list of student, call the list 'students'
- in this way a single item in the list will be the singular name ('color'), the entire list will be plural ('colors') - **Iteration pattern**

# Defining a list

A list is a container which holds comma-separated values (items or elements) between square brackets where Items or elements don't need all have the same type.

- elements in a list have an index

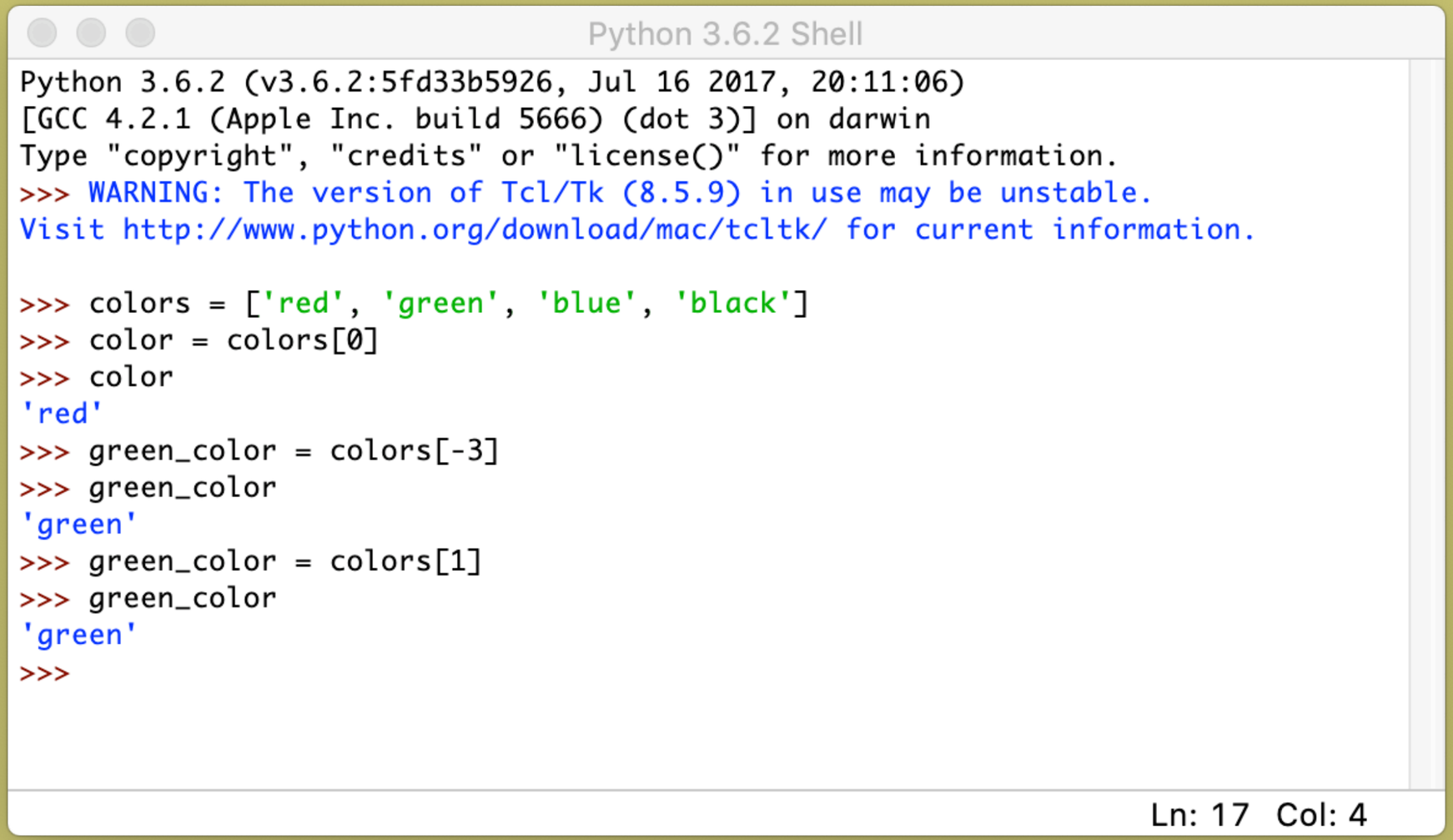
- list are zero-index: the first element in a list has 0 as index

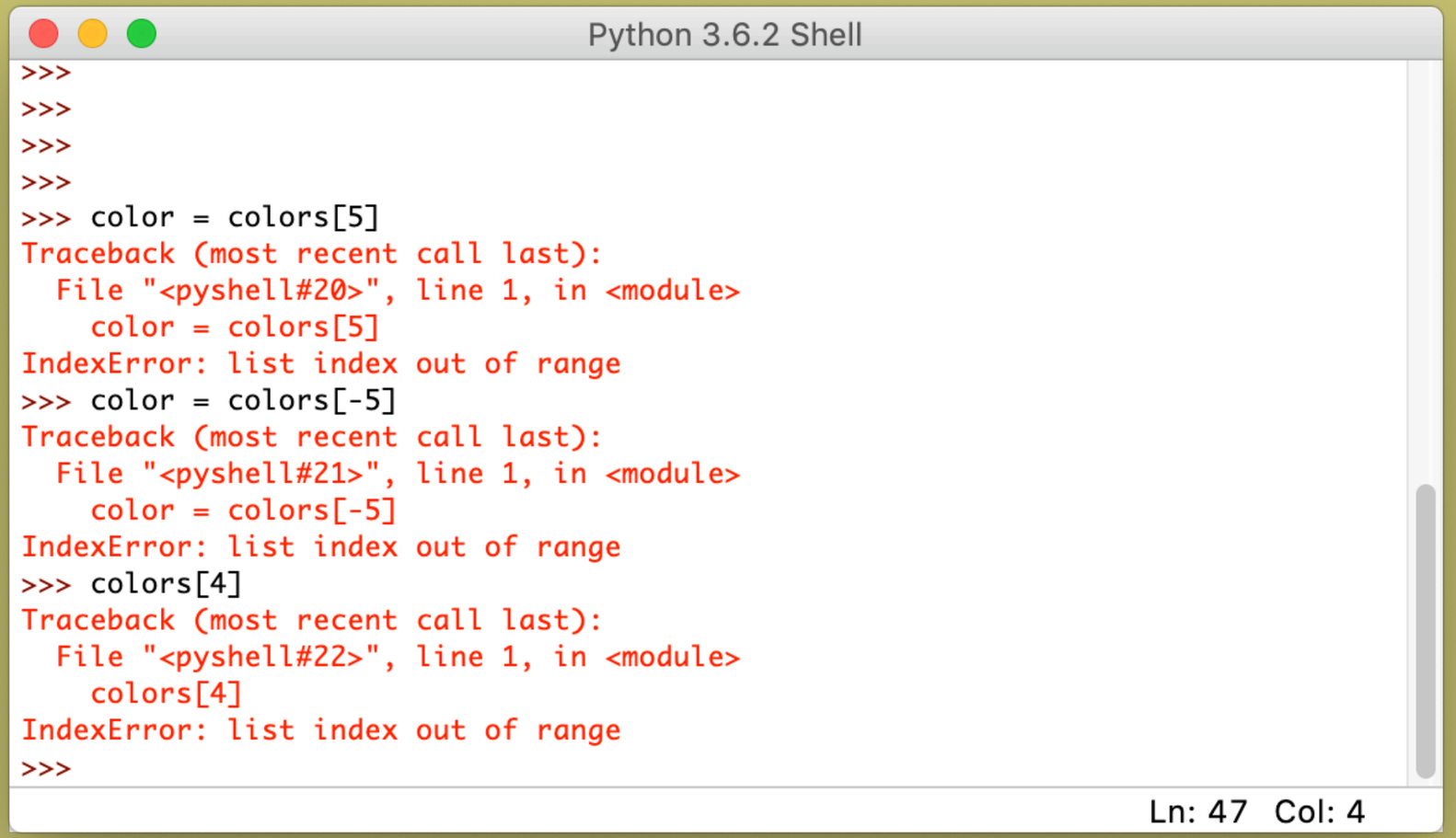
# List indexes

- index counts elements in a list
- if an index has a positive value, it counts from the beginning
- if an index has a negavite value, it counts backward
- you can get the element of a list by its index in square brackets

```
colors = ['red', 'green', 'blue', 'black']
```

| item | red | green | blue | black |
|---|---|---|---|---|
| index (from left) | 0 | 1 | 2 | 3 |
| index (from right) | -4 | -3 | -2 | -1 |


Python 3.6.2 Shell
Python 3.6.2 (v3.6.2:5fd33b5926, Jul 16 2017, 20:11:06)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.
>>> colors = ['red', 'green', 'blue', 'black']
>>> color = colors[0]
>>> color
'red'
>>> green_color = colors[-3]
>>> green_color
'green'
>>> green_color = colors[1]
>>> green_color
'green'
>>>
Ln: 17 Col: 4


Python 3.6.2 Shell
>>>
>>>
>>>
>>>
>>> color = colors[5]
Traceback (most recent call last):
File "<pyshell#20>", line 1, in <module>
color = colors[5]
IndexError: list index out of range
>>> color = colors[-5]
Traceback (most recent call last):
File "<pyshell#21>", line 1, in <module>
color = colors[-5]
IndexError: list index out of range
>>> colors[4]
Traceback (most recent call last):
File "<pyshell#22>", line 1, in <module>
colors[4]
IndexError: list index out of range
>>>
Ln: 47 Col: 4

# Common list operations

## Edit element in a list

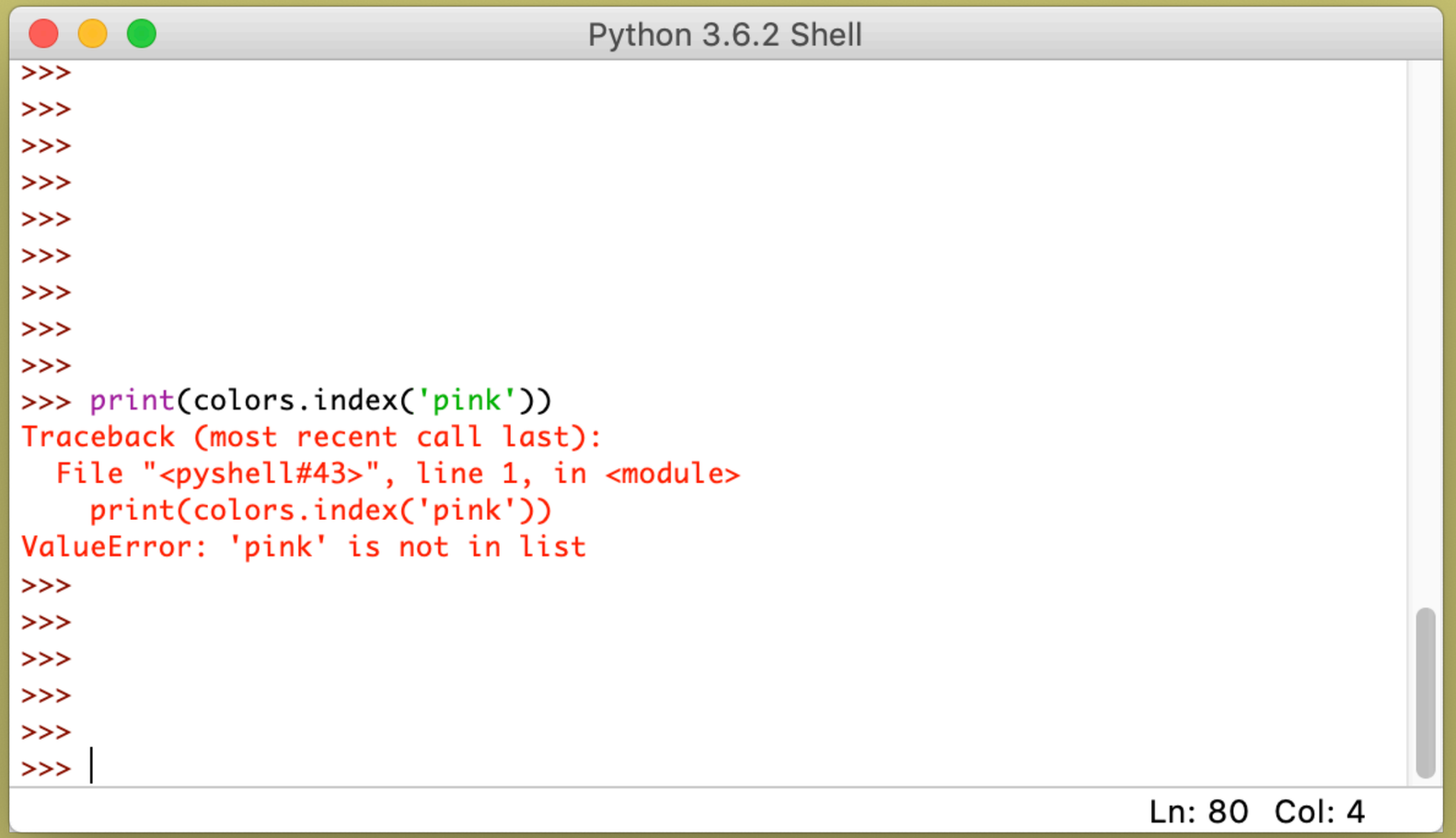
```
colors[0] = 'orange'

print(colors)
['orange', 'green', 'blue', 'black']
```

# Common list operations

## get the position of an element in list

```
colors = ['red', 'green', 'blue', 'black']
print(colors.index('green'))
1
```


Python 3.6.2 Shell
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>> print(colors.index('pink'))
Traceback (most recent call last):
File "<pyshell#43>", line 1, in <module>
print(colors.index('pink'))
ValueError: 'pink' is not in list
>>>
>>>
>>>
>>>
>>>
>>>
Ln: 80 Col: 4

# Common list operations

## test if an element is in a list

```
colors = ['red', 'green', 'blue', 'black']

print('red' in colors)
True
print('orange' in colors)
False
```

# Common list operations

## Adding item to the end of a list

```
colors = ['red', 'green', 'blue', 'black']

colors.append('orange')
print(colors)
['red', 'green', 'blue', 'black', 'orange']
```

# Common list operations

## Insert item into a list

we specify the position and everythings from that point is shifted one position to the rigth

```
colors = ['red', 'green', 'blue', 'black']

colors.insert(1, 'orange')
print(colors)
['red', 'orange', 'green', 'blue', 'black']
```

# Common list operations

## removing item by position

we specify the position and everythings from that point is shifted one position to the rigth

```
colors = ['red', 'green', 'blue', 'black']

del colors[1]
print(colors)
['red', 'blue', 'black']
```

# Common list operations

## removing item by value

remove **only the first** macthing value in a list

```
colors = ['red', 'green', 'blue', 'black']

colors.remove('green')
['red', 'blue', 'black']
```

# Common list operations

## popping item from a list

- remove the last item from the list (with no parameter)
- **always** it returns the popped item

```
colors = ['red', 'green', 'blue', 'black']

last_item = colors.pop()
print(last_item)
black
print(colors)
['red', 'green', 'blue']
```

# Common list operations

## popping item from a list

- or in a specific position
- **always** it returns the popped item

```
colors = ['red', 'green', 'blue', 'black']

item = colors.pop(1)
print(item)
green
print(colors)
['red', 'blue', 'black']
```

# Common list operations

## number of elements of a list

```
colors = ['red', 'green', 'blue', 'black']

number_item = len(colors)
print(number_item)
4
```

# Numerical lists

- a list composed by only numbers
- it's not special
- you can use some special function

The range() function is a shortcut to create a numerical list ordered

range(1,11): [1,2,3,4,5,6,7,8,9,10]

# range() function

The range() function returns a list of consecutive integers. The function has one, two or three parameters where last two parameters are optional.

- range(a): Generates a sequence of numbers from 0 to a, excluding a, incrementing by 1.

# range() function

- range(a, b): Generates a sequence of numbers from a to b excluding b, incrementing by 1.
- range(a,b,c): Generates a sequence of numbers from a to b excluding b, incrementing by c.

# Numerical lists

## min()

```
ages = [22, 38, 33, 50, 9, 16, 28, 11]

youngest = min(ages)
print(youngest)
9
```

# Numerical lists

## max()

```
ages = [22, 38, 33, 50, 9, 16, 28, 11]

oldest = max(ages)
print(oldest)
50
```

# Numerical lists

## sum()

```
ages = [22, 38, 33, 50, 9, 16, 28, 11]

total = sum(ages)
print(total)
207
```

# exercise

- calculate the ages' average
- add 3 different ages
- remove the last item
- remove age 22
- recalculate the ages'a average

# exercise

- Store the values 'python', 'c', and 'java' in a list. Print each of these values out, using their position in the list.

# exercise

- Print a statement about each of these values, using their position in the list.Your statement could simply be, 'A nice programming language is *value*.'

- Think of something you can store in a list. Make a list with three or four items, and then print a message that includes at least one item from your list. Your sentence could be as simple as, "One item in my list is a ____."

# Lists and Looping

## accessing all elements in a list

- we use a loop to access all the elements in a list, no matters how many elements the list was composed
- we are able to "use" all the elements in the list with 3 line of code
- **A loop** is a block of code that repeats itself until it runs out of items to work with or until a certain condition is met

```
colors = ['red', 'green', 'blue', 'black']

for color in colors:
    print(color)
red
green
blue
black
```

- the keyword "for" tells Python to get ready to use a loop
- the variable color is a placeholder variable. In thi variable python will place each item in the list into, one at a time
- when there are no more items in the list, the loop will end

# for loop and range function

```
for a in range(4):
  print(a)
```

```
0
1
2
3
```

# Enumerating a list

If you want to enumerate a list, you need to add an index variable to hold the current index

```
colors = ['red', 'green', 'blue', 'black']

for index, color in enumerate(colors):
    place = str(index)
    print("place: " + place + " color: " + color)
place: 0 color: red
place: 1 color: green
place: 2 color: blue
place: 3 color: black
```

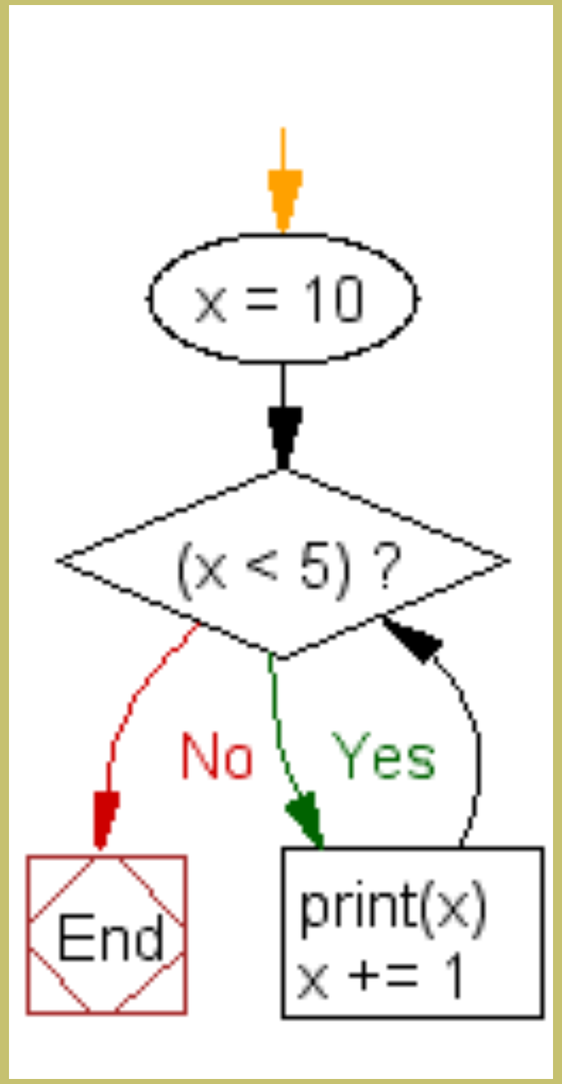
# Lists and Looping

## FOR Special keywords

Python allows two **keywords** to be used within a `for` loop: **break** and **continue**.

The two keywords have two **different** meanings:

- **Break** used to *immediatly break the loop and exit!*
- **Continue** used to skip to the **next** iteration step!

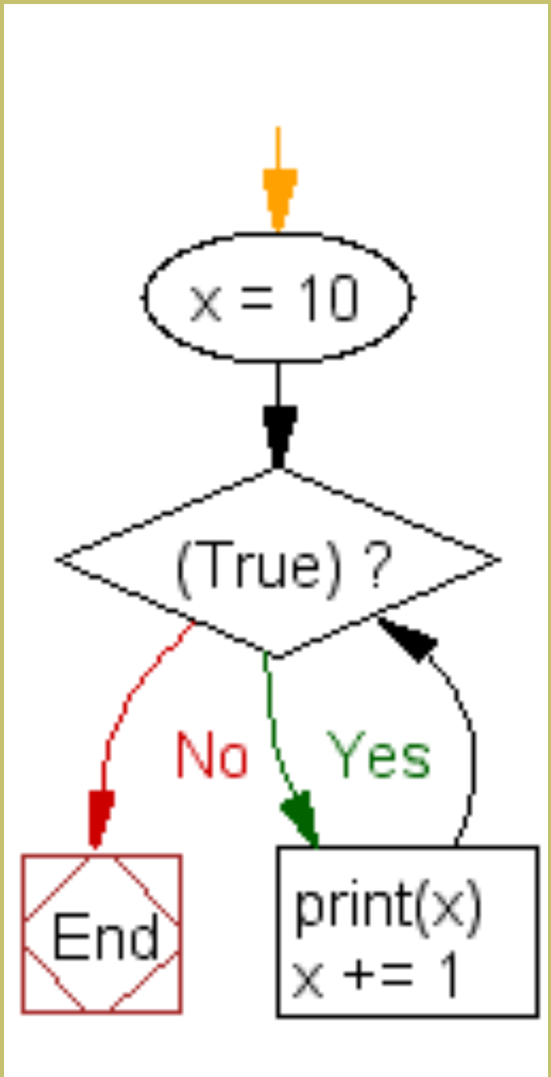



# While loop

In Python, also the while loops are permitted

The while loop runs as long as the conditio evaluates to True and execute the code block

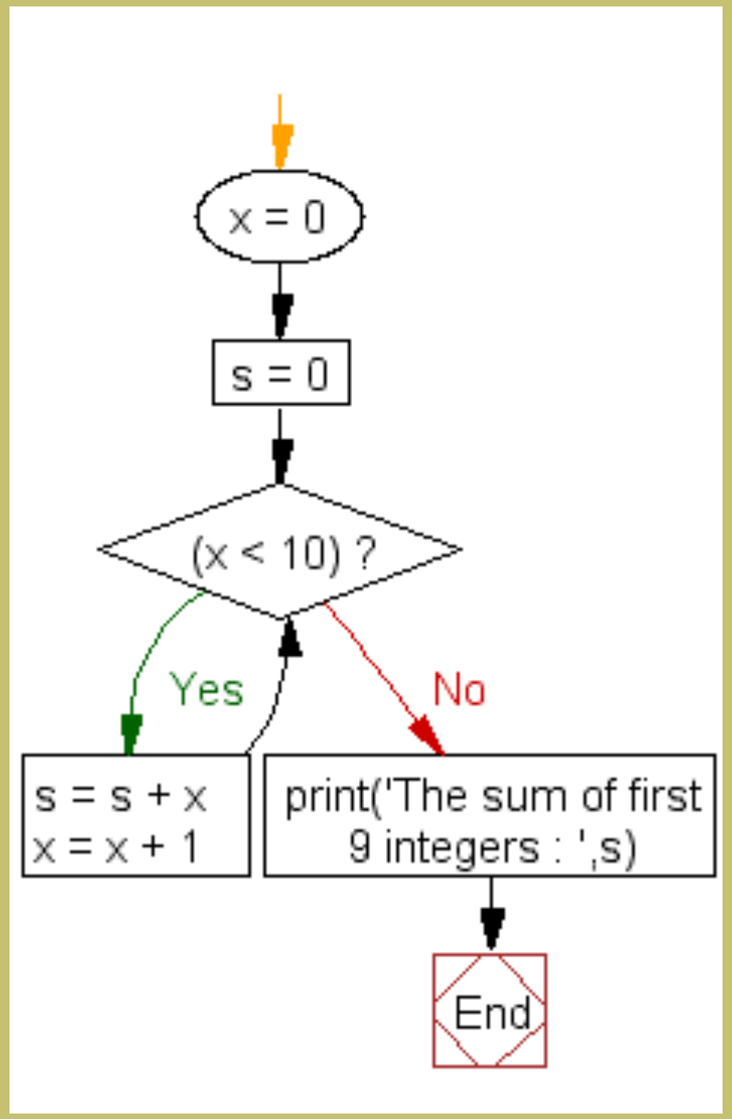
```
x = 0;
while (x < 5):
      print(x)
      x += 1
```




# While loop

## infinite loop

```
x = 10;
while (True):
      print(x)
      x += 1
```




# While loop

## else statement

`else` statement is executed when the condition is false

```
x = 0;
s = 0
while (x < 10):
      s = s + x
      x = x + 1
else :
      print('The sum of first 9 integers : ',s)
```

# exercise

- Repeat *First List*, but this time use a loop to print out each value in the list.
- Reapet *First List*, but this time use a loop to print out your statements. Make sure you are writing the same sentence for all values in your list
- Repeat *First List*, but this time use a loop to print out your message for each item in your list. Again, if you came up with different messages for each value in your list, decide on one message to repeat for each value in your list.

# Strings as Lists

- Now that you have some familiarity with lists, we can take a second look at strings.
- A string is really a list of characters, so many of the concepts from working with lists behave the same with strings.

# Strings as Lists

```
message = "Hello Lists!"
message_list = list(message)
print(message_list)
['H', 'e', 'l', 'l', 'o', ' ', 'L', 'i', 's', 't', 's', '!']
```

```
for letter in message:
    print(letter)
```

```
H
e
l
l
o

L
i
s
t
s
!
```

# Finding substring

A substring is a series of characters that appears in a string

```
>>> message = "I like cats and dogs"
>>> dog_present = 'dog' in message
>>> print(dog_present)
True
```

```
>>> message = "I like cats and dogs, but I prefer cats"
>>> first_cat_index = message.find('cats')
>>> print(first_cat_index)
7
>>> last_cat_index = message.rfind("cats")
>>> print(last_cat_index)
35
```

# Replace substrings

You can use replace() function to replace any substring with another substring.

```
>>> message = "I like cats and dogs, but I prefer cats"
>>> new_message = message.replace("cats","snakes")
>>> print(new_message)
I like snakes and dogs, but I prefer snakes
```

# Counting substrings

```
>>> number_of_cats = message.count('cats')
>>> print(number_of_cats)
2
```

# Splitting strings

```
>>> words = message.split(' ')
>>> print(words)
['I', 'like', 'cats', 'and', 'dogs,', 'but', 'I', 'prefer', 'cats']
```

# Slicing a list / 1

A subsets of a list are called slices

```
words = ['I', 'like', 'cats', 'and', 'dogs,', 'but', 'I', 'prefer', 'cats']
slice = words[0:3]
>>> for word in slice:
...     print(word)
...
I
like
cats
```

# Slicing a list /2

```
words = ['I', 'like', 'cats', 'and', 'dogs,', 'but', 'I', 'prefer', 'cats']
slice = words[2:5]
>>> slice = words[2:5]
>>> for word in slice:
...     print(word)
...
cats
and
dogs,
```

# Slicing a list /3

```
words = ['I', 'like', 'cats', 'and', 'dogs,', 'but', 'I', 'prefer', 'cats']
# from 4rd to the end
>>> slice = words[4:]
>>> for word in slice:
...     print(word)
dogs,
but
I
prefer
cats
```

# Slicing strings

```
>>> message = "I like cats and dogs, but I prefer cats"
>>> first_char = message[0]
>>> last_char = message[-1]
>>> print(first_char, last_char)
I s
>>> first_three = message[:3]
>>> last_three = message[-3:]
>>> print(first_three, last_three)
I l ats
```

# Exercise

Store the first 10 square numbers in a list.

Make an empty list that will hold our square numbers

```
>>> squares = []
>>> for number in range(1,11):
...     new_square = number**2
...     squares.append(new_square)
...
>>> for square in squares:
...     print(square)
...
1
4
9
16
25
36
49
64
81
100
```

# Exercise

- Make a list of the first ten multiples of ten (10, 20, 30...90, 100). Print out your list.
- Make a list of the first ten cubes(1, 8, 27...) using a list. Print them out
- Store five names in a list. Make a second list that adds the phrase "is awesome!" to each name. Print out the new list

# Tuples

- Tuples are basically lists that can never be changed
  - Tuples are not dynamic
  - you cannot modify any element
- Lists are mutable objects and tuple are immutable objects

# Defining Tuples

as the lists but use (), non []

```
colors = ('red', 'green', 'blue')
>>> print(colors[0])
red
>>> for color in colors:
...     print("- " + color)
...
- red
- green
- blue
```

# Adding an element to a Tuple

If you try to add something to a tuple, you will get an error:

```
colors = ('red', 'green', 'blue')
colors.append('purple')
```

```
---------------------------------------------------------------------
--
AttributeError                            Traceback (most recent call las
t)
<ipython-input-37-ed1dbff53ab2> in <module>()
      1 colors = ('red', 'green', 'blue')
----> 2 colors.append('purple')

AttributeError: 'tuple' object has no attribute 'append'
```

# Sets

- unordered collections of distinct objects
- You define a set just like you define a list, but you have to use **braces** "{}" instead of square brackets "[]"
- you can access individual elements (just like you can with a list and tuple)

```
>>> shapes = {"square", "triangle", "circle"}
>>> for shape in shapes:
...     print(shape)
...
circle
square
triangle
>>> shapes.add("polygon")
>>> shapes
{'polygon', 'circle', 'square', 'triangle'}
>>> shapes2 = set(["circle", "triangle", "hexagon"])
{'hexagon', 'circle', 'triangle'}
```

```
shapes = {'polygon', 'circle', 'square', 'triangle'}
shapes2 = {'hexagon', 'circle', 'triangle'}
>>> shapes.intersection(shapes2)
{'circle', 'triangle'}
>>> shapes.difference(shapes2)
{'polygon', 'square'}
>>> shapes.union(shapes2)
{'hexagon', 'circle', 'polygon', 'square', 'triangle'}
```

# Exercise

**3 is a Crowd**

- Make a list of names that includes at least 4 people
- Write an if test that prints a message about the room being crowded if there are more three people in your list
- Modify your list so that there are only two people in it. Use one of the methods for removing people from the list
- Run your if test again.

# Exercise

- Save your program from Three is Crowd under a new name
- add an else statement to ypur tests. If the else statement is run, have it print a message that the room is not very crowded

# Exercise

**Six is a Mob**

- save the previous program with a new name
- add some nemes to your list, so that there are at least six people in the list
- modify your tests so that:
  - more 5 people: print a message about there being a mob in the room
  - 3-5 pepole: print a message about there being crowded
  - 1 or 2 people: print a message about the room not being crowded
  - no people: pront a message about the room being empty