Lecture 4 – Compound data types, Traversals
https://cw.fel.cvut.cz/wiki/courses/be5b33prg/start

Tomas Jenicek
Czech Technical University in Prague,
Faculty of Electrical Engineering, Dept. of Cybernetics,
Center for Machine Perception
http://cmp.felk.cvut.cz/~jenicto2/
tomas.jenicek@fel.cvut.cz
```python
x = 10
print(type(x))

y = x
if (id(x)==id(y)):
    print("x and y refer to the same object")

x = x + 1
if (id(x) != id(y)):
    print("x and y refer to DIFFERENT objects!")

z = 10
if (id(y)==id(z)):
    print("y and z point to the SAME memory!!")
else:
    print("y and z point DIFFERENT objects!")
```

Output Window:
- `<class 'int'>`
- x and y refer to the same object
- x and y refer to DIFFERENT objects!
- y and z point to the SAME memory!!

Source: [https://www.youtube.com/watch?v=arxWaw-E8QQ&t=1s](https://www.youtube.com/watch?v=arxWaw-E8QQ&t=1s)
x = 10
print(type(x))

y = x
if (id(x)==id(y)):
    print("x and y refer to the same object")

x = x + 1
if (id(x) != id(y)):
    print("x and y refer to DIFFERENT objects!")

z = 10
if (id(y)==id(z)):
    print("y and z point to the SAME memory!!")
else:
    print("y and z point DIFFERENT objects!")

z = Car() #some user defined class
print(type(z))

<class 'int'>  
Output Window

x and y refer to the same object
x and y refer to DIFFERENT objects!
y and z point to the SAME memory!!
<class '__main__.Car'>

source https://www.youtube.com/watch?v=arxWaw-E8QQ&t=1s

Everything is object in Python

Python is a dynamically typed language
```python
#!/usr/bin/env python

def f1(x):
    x *= 2
    y = f2(x)
    return y

def f2(x):
    x += 1
    return x

if __name__ == '__main__':
    y = 5
    z = f1(y)
    print(z)
```

source: [https://www.youtube.com/watch?v=arxWaw-E8QQ&t=1s](https://www.youtube.com/watch?v=arxWaw-E8QQ&t=1s)
## MORE ABOUT PYTHON

<table>
<thead>
<tr>
<th>Statement</th>
<th>Python</th>
<th>JAVA / C</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 10</td>
<td>int x = 10;</td>
<td></td>
</tr>
<tr>
<td>What is 10?</td>
<td>An Object created on heap memory.</td>
<td>A primitive data stored in 4 byte</td>
</tr>
<tr>
<td>What does x contain?</td>
<td>Reference to Object 10</td>
<td>Memory location where 10 is stored</td>
</tr>
<tr>
<td>x = x + 1</td>
<td>x starts referring to a new object whose value is 11</td>
<td>x continues to point to the same memory, with value equal to 11</td>
</tr>
<tr>
<td>x = 10</td>
<td>Both x and y will refer to the same object.</td>
<td>x and y are two variables pointing to different memory locations.</td>
</tr>
<tr>
<td>y = 10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

source [https://www.youtube.com/watch?v=arxWaw-E8QQ&t=1s](https://www.youtube.com/watch?v=arxWaw-E8QQ&t=1s)
• Everything in Python is **object**
• Python is **dynamically typed** language
  (*type changes with reference*)
• The methods and variables are created on the **stack memory**
• The objects and instances are created on the **heap memory**
• New **stack frame** is created on invocation of a function / method and references are assigned & counted
• Stack frames are destroyed as soon as the function / method returns
• Mechanism to clean up the dead objects is **Garbage collector**
  (*algorithm used is Reference Counting and immediate object removal if count == 0*)

source [https://www.youtube.com/watch?v=arxWaw-E8QQ&t=1s](https://www.youtube.com/watch?v=arxWaw-E8QQ&t=1s)
Automate **repetitive tasks** without errors

Repeated execution of a set of statements is called **iteration**

Already explored **for**, now explore **while**

Running through all items in a list is **traversing / traversal**

```python
def mysum(xs):
    """Sum all the numbers in the list xs, and return the total.""
    running_total = 0
    for x in xs:
        running_total = running_total + x
    return running_total

# Add tests like these to your test suite ...
test(mysum([1, 2, 3, 4]) == 10)
test(mysum([1.25, 2.5, 1.75]) == 5.5)
test(mysum([1, -2, 3]) == 2)
test(mysum([]) == 0)
test(mysum(range(11)) == 55)  # 11 is not included in the list.
```

source [http://openbookproject.net/thinkcs/python/english3e/iteration.html](http://openbookproject.net/thinkcs/python/english3e/iteration.html)
The **while** statement has same meaning as in English

- Evaluate the condition (*at line 5*) either **False** or **True**.
- If the value is **False**, exit the **while** statement and continue execution at the next statement (*line 8 in this case*)
- If the value is **True**, execute each of the statements in the body (*lines 6 and 7*), then go back to the **while** statement

source [http://openbookproject.net/thinkcs/python/english3e/iteration.html](http://openbookproject.net/thinkcs/python/english3e/iteration.html)
The while loop is **more work** than the equivalent for loop.

Need to **manage the loop variable**: give it an **initial value**, test for completion, update it in the body to enable **termination**.

**Note**: `range` generates a list up to but excluding the last value.

```python
def sum_to(n):
    """Return the sum of 1+2+3 ... n ""
    ss = 0
    v = 1
    while v <= n:
        ss = ss + v
        v = v + 1
    return ss

# For your test suite
test(sum_to(4) == 10)
test(sum_to(1000) == 500500)
```
• Use a **for** loop if you know **how many times the loop will execute** (**definite iteration** — we know ahead some definite bounds for what is needed)

• Use a **for** to loop over **iterables** (to be explored in later classes) usually in combination with **in**

• Use **while** loop if you are required to **repeat computation until given condition is met**, and you cannot calculate in advance when this will happen (**indefinite iteration** — we do not know how many iterations will be needed)

source [http://openbookproject.net/thinkcs/python/english3e/iteration.html](http://openbookproject.net/thinkcs/python/english3e/iteration.html)
The **break** statement in Python **terminates the current loop** and resumes execution at the next statement.

The **continue** statement in Python **returns the control to the beginning of the current loop**.

The **continue** statement **rejects all the remaining statements in the current iteration of the loop**...
import random       # We cover random numbers in the
rng = random.Random() # modules chapter, so peek ahead.
number = rng.randrange(1, 1000) # Get random number between [1 and 1000).

guesses = 0
msg = ""

while True:
    guess = int(input(msg + "\nGuess my number between 1 and 1000: "))
    guesses += 1
    if guess > number:
        msg += str(guess) + " is too high.\n"
    elif guess < number:
        msg += str(guess) + " is too low.\n"
    else:
        break

• Guessing game
• This program makes use of the mathematical law of trichotomy (given real numbers a and b, exactly one of these three must be true: a > b, a < b, or a == b)

source http://openbookproject.net/thinkcs/python/english3e/iteration.html
So far built-in types like `int`, `float`, `bool`.

Compound data types: `strings`, `lists`, `dictionaries`, and `tuples` are different from the others because they are made up of smaller pieces (*characters in case of a string, items in case of a list*).

Types comprising smaller pieces are `compound data types`.

source [http://openbookproject.net/thinkcs/python/english3e/strings.html](http://openbookproject.net/thinkcs/python/english3e/strings.html)
• Example of paired data: *lists of names* and *lists of numbers*
• Advanced way of representing data: making a pair of things is as simple as putting them into parentheses (i.e. *tuples*)

```python

print(celebs)
print(len(celebs))

for (nm, yr) in celebs:
    if yr < 1980:
        print(nm)
```

source: [http://openbookproject.net/thinkcs/python/english3e/iteration.html](http://openbookproject.net/thinkcs/python/english3e/iteration.html)
NESTED DATA

```python
students = [
    ("John", ["CompSci", "Physics"]),
    ("Vusi", ["Maths", "CompSci", "Stats"]),
]

# Print all students with a count of their courses.
for (name, subjects) in students:
    print(name, "takes", len(subjects), "courses")

# Count how many students are taking CompSci
counter = 0
for (name, subjects) in students:
    for s in subjects:
        if s == "CompSci":
            counter += 1

print("The number of students taking CompSci is", counter)
```

- Data structure — a mechanism for grouping and organizing data to make it easier to use

source: [http://openbookproject.net/thinkcs/python/english3e/iteration.html](http://openbookproject.net/thinkcs/python/english3e/iteration.html)
The pair data example is an example of a **tuple**. Tuple groups any number of items into a **compound value**. Tuple is a **comma-separated sequence of values**.

Other languages often call it **records** *(some related information that belongs together)*. **Important**: strings and tuples are **immutable** *(once Python creates a tuple in memory, it cannot be changed)*. Elements of a tuple **cannot be modified**, new tuple holding **different information** should always be made instead!

Source: [http://openbookproject.net/thinkcs/python/english3e/tuples.html](http://openbookproject.net/thinkcs/python/english3e/tuples.html)
• Powerful **tuple assignment** (remember variable swapping?)
• Equivalent of **multiple assignment statements**
• Requirement: the number of **variables on the left** must match the **number of elements** in the tuple
• Tuple assignment is called tuple **packing / unpacking**

source [http://openbookproject.net/thinkcs/python/english3e/tuples.html](http://openbookproject.net/thinkcs/python/english3e/tuples.html)
• Use of tuples in functions as return value
• Function can always only return a single value, but by making that value a tuple, as many values can be packed together as is needed (e.g. find the mean and the standard deviation)
• Tuple items can themselves be other tuples (nested tuples)

• Heterogeneous data structure: can be composed of elements of different types (tuples, strings, lists)

source: http://openbookproject.net/thinkcs/python/english3e/tuples.html
• **Example**: `upper` is a method that can be invoked on any string object to create a new string, where all the characters are in uppercase

• `lower`, `capitalize`, `swapcase` ...

• **Use documentation & help!**
• Python uses **square brackets** to enclose the index – **indexing operator** `[]`
• The expression in brackets is called an **index**
• **Example**: *The expression `fruit[1]` selects character number 1 from `fruit`, and creates a new string containing just this one character*
• Computer scientists always start **counting from zero**!
• An index specifies a **member of an ordered collection** (in this case the collection of characters in the string)
• Index indicates **which one you want**, hence the name
• Index can be any **integer expression** (not only value)
• Use `enumerate` to visualize indices
• Note that indexing strings returns a string: Python has no special type for a single character (string of length = 1)
• Use `index` to extract elements from a list
• Use `len` to extract the **number of elements** (indexing from 0!)
• Negative indices count backward from the end of the string
• *The expression `fruit[-1]` yields the last letter*
• Traversals: **while** vs. **for** comparison again!

```python
>>> fruit = "banana"
>>> len(fruit)
6
1 2 1 2
`sz = len(fruit)`
`last = fruit[sz]`  # ERROR!
`sz = len(fruit)`
`last = fruit[sz-1]`

IndexError: string index out of range.
```
A substring of a string is obtained by taking a slice

Slice a list to refer to some sublist of the items in the list

The operator [n:m] returns the part of the string from the n’th character to the m’th character, including the first but excluding the last (indices pointing between the characters)

Slice operator [n:m] copies out the part of the paper between the n and m positions

Result of [n:m] will be of length (m-n)

source http://openbookproject.net/thinkcs/python/english3e/strings.html
If you **omit the first index** (before the colon), the slice **starts at the beginning** of the string (or list)

If you **omit the second index**, the slice **extends to the end** of the string (or list)

If you provide value for `n` that is bigger than the length of the string (or list), the slice will **take all the values** up to the end

No **“out of range” error** like the normal indexing operation

source [http://openbookproject.net/thinkcs/python/english3e/strings.html](http://openbookproject.net/thinkcs/python/english3e/strings.html)
• **Comparing strings**: strings are **sorted** in the alphabetical order (except that all uppercase letters come before the lowercase)

• Strings are **immutable** (existing string cannot be changed, new one should be created instead)
The `in` / `not in` operator tests for membership.

Method `index` is the opposite of the indexing operator: it takes a character (item in case of a list) and finds the index of the character / item (if not found then exception is raised).

Method `find` works for strings in a similar way (if the character is not found, the function returns -1).

```python
def find(strng, ch):
    
    Find and return the index of ch in strng.
    Return -1 if ch does not occur in strng.
    
    ix = 0
    while ix < len(strng):
        if strng[ix] == ch:
            return ix
        ix += 1
    return -1

test(find("Compsci", "p") == 3)
test(find("Compsci", "C") == 0)
test(find("Compsci", "i") == 6)
test(find("Compsci", "x") == -1)
```

Source: [http://openbookproject.net/thinkcs/python/english3e/strings.html](http://openbookproject.net/thinkcs/python/english3e/strings.html)
• The **split** method:

> it splits a single multi-word string into a list of individual words, removing all the whitespace between them (whitespace are: tabs, newlines, spaces)

• Explore the **join** method on your own!
The `format` method substitutes its arguments into the place holders (numbers are indexes of the arguments).

- Format specification — it is always introduced by the colon `:`
- Field is aligned to the `left <`, `center ^`, or `right >`
- Width allocated to the field within the result string
- Type conversion
- Specification of decimal places
  (`.2f` is useful for when rounding to two decimal places.)

source [http://openbookproject.net/thinkcs/python/english3e/strings.html](http://openbookproject.net/thinkcs/python/english3e/strings.html)
• A list is an ordered collection of values
• Values of a list are called its elements or items
• Similar to strings (ordered collections of characters) except that the elements of a list can be of any type

• Lists and strings — and other collections that maintain the order of their items — are called sequences
• List within list is said to be nested
• List with no elements is called an empty list, and is denoted []
• Expression evaluating to an integer can be used as an index
• Function `len` returns length of a list (number of its elements)
• Testing membership using `in` / `not in`
• Operators `+` (concatenation) and `*` (repetition)
Lists are **mutable** *(we can change list elements)*

- Use same **slicing principles** as for strings
- Use `del` to delete list elements

---

**source** [http://openbookproject.net/thinkcs/python/english3e/lists.html](http://openbookproject.net/thinkcs/python/english3e/lists.html)
- Variables `a` and `b` refer to string object with letters "banana"
- Use `is` operator or `id` function to find out the reference
- Strings are immutable
  
  ```python
  >>> a = [1, 2, 3]
  >>> b = [1, 2, 3]
  >>> a == b
  True
  >>> a is b
  False
  ```

  *Python optimizes resources by making two names that refer to the same string value refer to the same object*

- Not the case of lists: `a` and `b` have the same value (content) but do not refer to the same object

source: [http://openbookproject.net/thinkcs/python/english3e/lists.html](http://openbookproject.net/thinkcs/python/english3e/lists.html)
• If we assign one variable to another, both variables refer to the same object
• The same list has two different names we say that it is aliased (changes made with one alias affect the other)
• Recommendation: Avoid aliasing when you are working with mutable objects!
• If need to modify a list and keep a copy of the original use the slice operator (taking any slice of creates a new list)

source [http://openbookproject.net/thinkcs/python/english3e/lists.html](http://openbookproject.net/thinkcs/python/english3e/lists.html)
• Passing a list as an argument passes a reference to the list, not a copy or clone of the list!

• So parameter passing creates an alias!

source: http://openbookproject.net/thinkcs/python/english3e/lists.html
>>> mylist = []
>>> mylist.append(5)
>>> mylist.append(27)
>>> mylist.append(3)
>>> mylist.append(12)
>>> mylist
[5, 27, 3, 12]

>>> mylist.insert(1, 12)  # Insert 12 at pos 1, shift other items up
>>> mylist
[5, 12, 27, 3, 12]
>>> mylist.count(12)      # How many times is 12 in mylist?
2
>>> mylist.extend([5, 9, 5, 11])  # Put whole list onto end of mylist
>>> mylist
[5, 12, 27, 3, 12, 5, 9, 5, 11]
>>> mylist.index(9)       # Find index of first 9 in mylist
6
>>> mylist.reverse()
>>> mylist
[11, 5, 9, 5, 12, 3, 27, 12, 5]
>>> mylist.sort()
>>> mylist
[3, 5, 5, 5, 9, 11, 12, 12, 27]
>>> mylist.remove(12)     # Remove the first 12 in the list
>>> mylist
[3, 5, 5, 5, 9, 11, 12, 27]

Explore list methods on your own!
**Concept: pure functions vs. modifiers**

- Pure function does not produce **side effects**!
- Pure function communicates with the calling program only through parameters (it does not modify) and a return value
- *Do not alter the input parameters unless really necessary*
- Programs that use pure functions are faster to develop and less error-prone than programs that use modifiers
This lecture re-uses selected parts of the OPEN BOOK PROJECT

Learning with Python 3 (RLE)

http://openbookproject.net/thinkcs/python/english3e/index.html

available under GNU Free Documentation License Version 1.3

• Version date: October 2012
• by Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers (based on 2nd edition by Jeffrey Elkner, Allen B. Downey, and Chris Meyers)
• Source repository is at https://code.launchpad.net/~thinkcspy-rle-team/thinkcspy/thinkcspy3-rle
• For offline use, download a zip file of the html or a pdf version from http://www.ict.ru.ac.za/Resources/cspw/thinkcspy3/