Lecture 3 – Program structure, Functions
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
RECAP: LOOPS – FOR, WHILE

On each iteration or pass of the loop:
• Check to see if there are still more items to be processed
• If there are none left (the terminating condition of the loop) the loop has finished
• If there are items still to be processed, the loop variable is updated to refer to the next item in the list
• Program continues at the next statement after the loop body
• To explore: early break, or for – else loop, while loop

source http://openbookproject.net/thinkcs/python/english3e/hello_little_turtles.html
• Recommendation: early return / early break
• Special condition: FOR – ELSE
• Explore on your own: for, in, while, if, else, break, continue

### HIERARCHY OF TYPES

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td></td>
</tr>
<tr>
<td>- str</td>
<td>“look”</td>
</tr>
<tr>
<td>- int</td>
<td>3</td>
</tr>
<tr>
<td>- bool</td>
<td>True</td>
</tr>
<tr>
<td>- float</td>
<td>3.0</td>
</tr>
<tr>
<td>- NoneType</td>
<td>None</td>
</tr>
</tbody>
</table>

- Data types have hierarchy
- “Everything is an object”

1. **Global** definitions section
2. **Function definitions / classes definitions** section
3. **Sequence of instructions** section (here the main section)

source: https://developers.google.com/edu/python/introduction

Michal Reinštein, Czech Technical University in Prague
• When python interpreter runs a source file as main program, it sets `__name__` variable to have a value "__main__"
• If being imported from another module, `__name__` will be set to the module’s name

source [https://developers.google.com/edu/python/introduction](https://developers.google.com/edu/python/introduction)
• Use **import** to include functions / classes from other modules

---

You can find the documentation of all the Standard Library modules and packages at [http://docs.python.org/library](http://docs.python.org/library).

---

• sys — access to exit(), argv, stdin, stdout, ...
• re — regular expressions
• os — operating system interface, file system

---

source [https://developers.google.com/edu/python/introduction](https://developers.google.com/edu/python/introduction)
```python
#!/usr/bin/env python

# import modules used here. -- sys is a very standard one
import sys

# Gather our code in a main() function
def main():
    print('Hello there', sys.argv[0])
    # Command line args are in sys.argv[1], sys.argv[2] ...
    # sys.argv[0] is the script name itself and can be ignored
    for n in range(2, 10):
        print('n = ', n)
        for x in range(2, n):
            print('x = ', x)
            if n % x == 0:
                print(n, 'equals', x, '*', n // x)
                break
            else:
                # loop fell through without finding a factor
                print(n, 'is a prime number')

# Standard boilerplate to call the main() function to begin
if __name__ == '__main__':
    main()
```

Process finished with exit code 0
WHY FUNCTIONS?

• Organize program into **chunks** that match **how we think** about the problem

• Code **re-using** without copy-paste

• Enforcing logical **structure** into the code

• Easier **debugging**

• Code **readability**

source [http://openbookproject.net/thinkcs/python/english3e/functions.html](http://openbookproject.net/thinkcs/python/english3e/functions.html)
FUNCTION DEFINITION

- **Function** = named sequence of statements belonging together
- **Header line**: begins with a keyword `def`, ends with a colon `:`
- **Body**: one or more statements, each indented the same amount
- **Parameter list**: empty or any number of comma separated parameters (can have default value)
- Any **name** except for keywords and illegal identifiers
- Any **number of statements** inside the function, but **indented** from the `def` (standard indentation of four spaces)
- Function may or may not produce a result

source [http://openbookproject.net/thinkcs/python/english3e/functions.html](http://openbookproject.net/thinkcs/python/english3e/functions.html)
• Most functions require **arguments** (named arguments, default values)
• More than one argument: e.g. `pow(base, exponent)`
• Functions like **range**, **int**, **abs** all return values that can be used to build more complex expressions
• Function that returns value is called a **fruitful function**
• Opposite of a fruitful function is **void function** (procedure)
LOCAL VARIABLES

```
1 def final_amt(p, r, n, t):
2     a = p * (1 + r/n) ** (n*t)
3     return a
```

If we try to use `a`, outside the function, we’ll get an error:

```
>>> a
NameError: name 'a' is not defined
```

- When a variable is created inside a function, it is **local** and cannot be used outside (shadowing names)
- The variable `a` is local to `final_amt`
- Local variables only exist while the function is being executed — this is called variable **lifetime**
- Parameters are local and act like local variables
• Docstrings are meant for **documentation** (if the first thing after the function header is string then treated as docstring)
• Key way to **document** our functions
• Concept of abstraction (need to know the interface)
• Formed using **triple-quoted** strings
• Different from comments: retrievable by Python tools at **runtime** (comments are completely eliminated during parsing)
Functions such as `abs`, `pow`, `int`, `max`, `range`, produce results. Return statement of fruitful functions includes a return value. Temporary variables like `b` above make debugging easier.
Multiple return statements, one in each branch of conditional
Code after return is called **dead code**, or unreachable code
All Python functions return **None** whenever they do not return another value.

source [http://openbookproject.net/thinkcs/python/english3e/fruitful_functions.html](http://openbookproject.net/thinkcs/python/english3e/fruitful_functions.html)
Boolean functions are often used in conditional statements:

```python
def is_divisible(x, y):
    """ Test if x is exactly divisible by y """
    if x % y == 0:
        return True
    else:
        return False
```

- Functions that return Boolean values
- Give Boolean functions names that sound like yes/no questions, e.g. `is_divisible`
- Condition of the `if` statement is itself a **Boolean expression**
Functions hide complex computation behind a single command and capture abstraction of the problem.

Functions can simplify a program.

Creating a new function can make a program shorter by eliminating repetitive code.

source: http://openbookproject.net/thinkcs/python/english3e/functions.html
• https://docs.python.org/3.4/library/math.html

source http://openbookproject.net/thinkcs/python/english3e/functions.html
Flow of execution = order of statements execution (begins at the first statement of the program)

- Statements are executed one at a time, in order from top to bottom (but read the flow, not top to bottom!)
- Python evaluates expressions from left to right (during assignment right-hand side is evaluated first)
- Function calls are like a detour in the flow of execution
- We can define one function inside another
- Function definitions do not alter the flow of execution

Swapping variables

source: [http://docs.python.org/3/reference/expressions.html#evaluation-order](http://docs.python.org/3/reference/expressions.html#evaluation-order)
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!")

<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-E8QO&t=1s](https://www.youtube.com/watch?v=arxWaw-E8QO&t=1s)
```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!")

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

Michal Reinštein, Czech Technical University in Prague
```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>Python</td>
<td>int x = 10;</td>
<td></td>
</tr>
<tr>
<td>Java</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

|--------------------------|-------------------------------|-------------------------------|

<table>
<thead>
<tr>
<th>What is 10?</th>
<th>An Object created on heap memory.</th>
<th>A primitive data stored in 4 byte memory.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>What does x contain?</th>
<th>Reference to Object 10</th>
<th>Memory location where 10 is stored</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>x = x + 1</th>
<th>x starts referring to a new object whose value is 11</th>
<th>x continues to point to the same memory, with value equal to 11</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>x = 10</th>
<th>Both x and y will refer to the same object.</th>
<th>x and y are two variables pointing to different memory locations.</th>
</tr>
</thead>
</table>

source: [https://www.youtube.com/watch?v=arxWaw-E8QQ&t=1s](https://www.youtube.com/watch?v=arxWaw-E8QQ&t=1s)
• Return statement in the middle of a `for` loop – control immediately returns from the function

• **EXAMPLE:** Let us assume that we want a function which looks through a list of words. It should return the first 2-letter word. If there is not one, it should return “Nothing found”
Incremental development technique – avoid long debugging sessions by adding and testing only a small amount of code at a time.

EXAMPLE: We want to find the distance between two points, given by the coordinates \((x_1, y_1)\) and \((x_2, y_2)\). 

(Pythagorean theorem)

What are the inputs (parameters)?
What is the output (return value)?

source https://openbookproject.net/thinkcs/python/english3e/fruitful_functions.html
Define interface

```
def distance(x1, y1, x2, y2):
    return 0.0
```

Process parameters

```
def distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    return 0.0
```

Temporary variables

```
def distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    dsquared = dx*dx + dy*dy
    return 0.0
```

Return result

```
def distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    dsquared = dx*dx + dy*dy
    result = dsquared**0.5
    return result
```

source: [http://openbookproject.net/thinkcs/python/english3e/fruitful_functions.html](http://openbookproject.net/thinkcs/python/english3e/fruitful_functions.html)
• Start with a working **skeleton program** and make small **incremental changes** (analyze errors)
• Use **temporary variables** to refer to intermediate values for easy inspection
• Once the program is working, **explore options** and parameters
• Consolidate multiple statements to make **shorter code**, refactor for readability
These are the terms you should explore and know:

- Argument
- Header
- Body
- Docstring
- Flow of execution
- Frame
- Function
- Function call
- Function composition
- Function definition
- Fruitful function
- Header line
- Import statement
- Lifetime
- Local variable
- Parameter
- Refactor
- Stack diagram
- Traceback (stack trace)
- void function

Learning with Python 3 - chapter 4.8
http://openbookproject.net/thinkcs/python/english3e/functions.html

source http://openbookproject.net/thinkcs/python/english3e/functions.html
The formula for computing the final amount if one is earning compound interest is given on Wikipedia as

\[ A = P \left( 1 + \frac{r}{n} \right)^{nt} \]

Where,

- \( P \) = principal amount (initial investment)
- \( r \) = annual nominal interest rate (as a decimal)
- \( n \) = number of times the interest is compounded per year
- \( t \) = number of years

Write a Python program that assigns the principal amount of $10000 to variable \( P \), assign to \( n \) the value 12, and assign to \( r \) the interest rate of 8%. Then have the program prompt the user for the number of years \( t \) that the money will be compounded for. Calculate and print the final amount after \( t \) years.

source [http://openbookproject.net/thinkcs/python/english3e/functions.html](http://openbookproject.net/thinkcs/python/english3e/functions.html)
EXAMPLE

\[ A = P \left(1 + \frac{r}{n}\right)^{nt} \]

Where,
- \( P \) = principal amount (initial investment)
- \( r \) = annual nominal interest rate (as a decimal)
- \( n \) = number of times the interest is compounded per year
- \( t \) = number of years

• Will be evaluated and returned to the caller as the “fruit”
• Input **prompt** from user (**type conversion** from string to float)
• Arguments for **8%** interest, compounded **12** times per year, for **5** years period
• **NOTE**: It is as if \( p = \text{toInvest} \) is executed when \( \text{final_amt} \) is called (variable name in the caller does not matter, in \( \text{final_amt} \) the name is \( p \) with **lifetime** until return)

```python
def final_amt(p, r, n, t):
    
    """
    Apply the compound interest formula to \( p \)
    to produce the final amount.
    """
    a = p * (1 + r/n)**(n*t)
    return a # This is new, and makes the function fruitful.

# now that we have the function above, let us call it.
toInvest = float(input("How much do you want to invest?"))
fnl = final_amt(toInvest, 0.08, 12, 5)
print("At the end of the period you'll have", fnl)
```

source [http://openbookproject.net/thinkcs/python/english3e/functions.html](http://openbookproject.net/thinkcs/python/english3e/functions.html)
EXAMPLE

\[ A = P \left(1 + \frac{r}{n}\right)^{nt} \]

Where,
- \( P \) = principal amount (initial investment)
- \( r \) = annual nominal interest rate (as a decimal)
- \( n \) = number of times the interest is compounded per year
- \( t \) = number of years

```python
def final_amt(p, r, n, t):
    """
    Apply the compound interest formula to \( p \) to produce the final amount.
    """
    a = p * \left(1 + \frac{r}{n}\right)^{(n*t)}
    return a  # This is new, and makes the function fruitful.

# now that we have the function above, let us call it.
toInvest = float(input("How much do you want to invest?"))
fnl = final_amt(toInvest, 0.08, 12, 5)
print("At the end of the period you'll have", fnl)
```

```python
def final_amt_v2(principalAmount, nominalPercentageRate, numTimesPerYear, years):
    a = principalAmount * \left(1 + \frac{\text{nominalPercentageRate}}{\text{numTimesPerYear}}\right)^{\text{numTimesPerYear}*\text{years}}
    return a

def final_amt_v3(amt, rate, compounded, years):
    a = amt * \left(1 + \frac{\text{rate}}{\text{compounded}}\right)^{\text{compounded}*\text{years}}
    return a
```

source: [http://openbookproject.net/thinkcs/python/english3e/functions.html](http://openbookproject.net/thinkcs/python/english3e/functions.html)
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/