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
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
RECAP: LOOPS & CONDITIONS

```python
for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print(n, 'equals', x, '*', n//x)
            break
```

```python
for n in range(2, 10):
    for x in range(2, n):
        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')
```

- Recommendation: **early return / early break**
- Special condition: **FOR – ELSE**
- Explore on your own: **for, in, while, if, else, break, continue**

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](https://developers.google.com/edu/python/introduction)

07/10/21
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
• 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).
```python
#!/usr/bin/env python

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):
            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()
```

Hello there /Users/michalreinstein/Dropbox/University/2023-24/Programming/Modelling/Python

Process finished with exit code 0
• **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)
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)

Michal Reinštein, Czech Technical University in Prague
• [https://docs.python.org/3.4/library/math.html](https://docs.python.org/3.4/library/math.html)
• 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)
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

(source: [http://docs.python.org/3/reference/expressions.html#evaluation-order](http://docs.python.org/3/reference/expressions.html#evaluation-order))
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](http://openbookproject.net/thinkcs/python/english3e/functions.html)
FUNCTIONS CALLING FUNCTIONS

```python
# /usr/bin/env python

def compute_area_rectangle(height, width):
    rtype: float
    rtype: float
    rtype: area of rectangle (m^2)
    # use assert as function guard
    assert height >= 0 and width >= 0, 'Length cannot be negative'
    return height * width

def compute_area_square(side):
    return compute_area_rectangle(side, side)

if __name__ == '__main__':
    square_side_length = float(input('Input square side length (m): '))
    print(compute_area_square(square_side_length))
```

```
/opt/local/bin/python3.6 '/Users/michalreinstein/Disk Google/TEACHING/BE5B33PRG_2017/examples/example_02.py'
Input square side length (m): 17
Traceback (most recent call last):
  File '/Users/michalreinstein/Disk Google/TEACHING/BE5B33PRG_2017/examples/example_02.py', line 25, in <module>
    print(compute_area_square(square_side_length))
  File '/Users/michalreinstein/Disk Google/TEACHING/BE5B33PRG_2017/examples/example_02.py', line 20, in compute_area_square
    return compute_area_rectangle(side_length, side_length)
  File '/Users/michalreinstein/Disk Google/TEACHING/BE5B33PRG_2017/examples/example_02.py', line 15, in compute_area_rectangle
    assert height >= 0 and width >= 0, 'Length cannot be negative'
AssertionError: Length cannot be negative
```

Process finished with exit code 1

source [http://openbookproject.net/thinkcs/python/english3e/functions.html](http://openbookproject.net/thinkcs/python/english3e/functions.html)
\begin{verbatim}
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!")
\end{verbatim}

\textbf{Output Window}

\begin{itemize}
\item `<class 'int'>`
\item \textbf{x and y} refer to the same object
\item \textbf{x and y} refer to DIFFERENT objects!
\item \textbf{y and z} point to the SAME memory!!
\end{itemize}

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

y = x
if (x is 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'>

Everything is object in Python
Python is a dynamically typed language

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

Michal Reinštein, Czech Technical University in Prague
# More about Python

<table>
<thead>
<tr>
<th>Python</th>
<th>JAVA / C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement</td>
<td>int x = 10;</td>
</tr>
<tr>
<td>Data type declaration</td>
<td>Not needed. Dynamically typed.</td>
</tr>
<tr>
<td><strong>What is 10?</strong></td>
<td>An Object created on heap memory.</td>
</tr>
<tr>
<td><strong>What does x contain?</strong></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>
</tr>
<tr>
<td>x = 10</td>
<td>x and y are two variables pointing to different memory locations.</td>
</tr>
<tr>
<td>y = 10</td>
<td></td>
</tr>
</tbody>
</table>

Source: [https://www.youtube.com/watch?v=arxWaw-E8QO&t=1s](https://www.youtube.com/watch?v=arxWaw-E8QO&t=1s)
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)

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

source [http://openbookproject.net/thinkcs/python/english3e/fruitful_functions.html](http://openbookproject.net/thinkcs/python/english3e/fruitful_functions.html)
FRUITFUL FUNCTIONS

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)
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
• 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)

\[
distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}
\]

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

source [http://openbookproject.net/thinkcs/python/english3e/fruitful_functions.html](http://openbookproject.net/thinkcs/python/english3e/fruitful_functions.html)
Define interface

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

Process parameters

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

Temporary variables

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

Return result

```python
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

```python
import math
def distance(x1, y1, x2, y2):
    return math.sqrt((x2-x1)**2 + (y2-y1)**2)

>>> distance(1, 2, 4, 6)
5.0
```
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.
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

```
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)^{\left(n\cdot t\right)}
    return a

# 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)
```

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

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

```python
def final_amt_v2(principalAmount, nominalPercentageRate, numTimesPerYear, years):
    a = principalAmount * (1 + nominalPercentageRate / numTimesPerYear) ** (numTimesPerYear*years)
    return a

def final_amt_v3(amt, rate, compounded, years):
    a = amt * (1 + rate/compounded)**(compounded*years)
    return a
```

source 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/