

# ROAR III

*Ricerca Operativa Applicazioni Reali*

---

Alessandro Gobbi   Alice Raffaele   Gabriella Colajanni   Eugenia Taranto

IIS Antonietti, Iseo (BS)

24 ottobre 2022

# Introduzione

---

# Chi siamo e i nostri contatti



Alessandro Gobbi (UniBS)  
*alessandro.gobbi@unibs.it*



Alice Raffaele (Univr)  
*alice.raffaele@univr.it*



Gabriella Colajanni (Unict)  
*gabriella.colajanni@unict.it*



Eugenia Taranto (Unict)  
*eugenia.taranto@unict.it*

# Riassunto – ROAR: perché questo nome?

Perché... la matematica è dappertutto! Ha un sacco di **applicazioni reali** e ve lo dimostreremo con:

1. casi di studio della vostra quotidianità;
2. casi di studio industriali.

Metodologia didattica: uso di **progetti**

- assegnamento di un compito da affrontare a gruppi;
- presentazione finale.

## Sei incontri con noi:

1. 15 marzo (5 ore – Introduzione, problema dello zaino, modelli a due variabili, presentazione compito autentico)
2. 27 marzo (2,5 ore – Modelli a due variabili e risoluzione grafica)
3. 12 aprile (5 ore – Risoluzione grafica e introduzione a Excel Solver)
4. 24 aprile (2,5 ore – Uso di Excel Solver)
5. 06 maggio (2 ore – Consulto per compiti autentici)
6. 13 maggio (2 ore – Esposizione progetti)

## Sette incontri con noi:

1. 17 gennaio (4 ore – Ripasso, introduzione al nuovo argomento, social networks)
2. 29 gennaio (4 ore – Problema del trasporto e algoritmo di Kruskal)
3. 05 febbraio (4 ore – Algoritmo di Dijkstra)
4. 11 febbraio (6 ore – Workshop all'Università di Brescia)
5. 12 marzo (4 ore – Problema del Postino Rurale)
6. 21 marzo (4 ore – Problema del commesso viaggiatore e presentazione progetto finale)
7. 23 aprile (4 ore – Esposizione progetto e questionario finale)

## 6 nuovi incontri con noi:

1. 24 ottobre (4 ore – Ripasso, introduzione al nuovo argomento, nozioni base di Python, la libreria PuLP e Google Colab)
2. 05 novembre (4 ore – Implementazione e risoluzione con Python+PuLP di problemi di Programmazione Lineare Mista Intera – Parte 1)
3. 14 novembre (4 ore – Implementazione e risoluzione con Python+PuLP di problemi di Programmazione Lineare Mista Intera – Parte 2 + Presentazione del progetto finale)
4. 26 novembre (4 ore – Lavoro di gruppo cooperativo sul progetto finale + Seminario aziendale)
5. 03 dicembre (4 ore – Lavoro di gruppo e cooperativo sul progetto finale)
6. Metà gennaio (2 ore – Esposizione progetto e questionari finali)

# Parte 1:

## Python e PuLP

---



# Il linguaggio di programmazione Python



*<https://www.python.org>*

- Linguaggio **ad alto livello** e **orientato agli oggetti**.
- Creato dall'informatico olandese **Guido van Rossum**:
  - 1991: Python 0.9.0
  - 2000: Python 2.0
  - 2008: Python 3.0
  - 2020: Python 2.7.18 viene dismesso
  - 2022: Python 3.10.7 (versione attuale)
- **Usi principali**: scripting, prototipazione e testing, analisi e visualizzazione di dati, didattica.

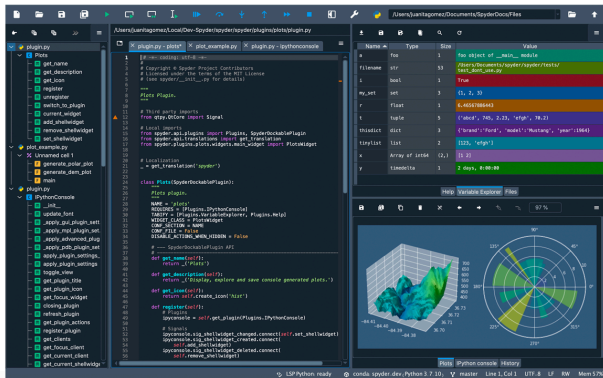
# Popolarità di Python



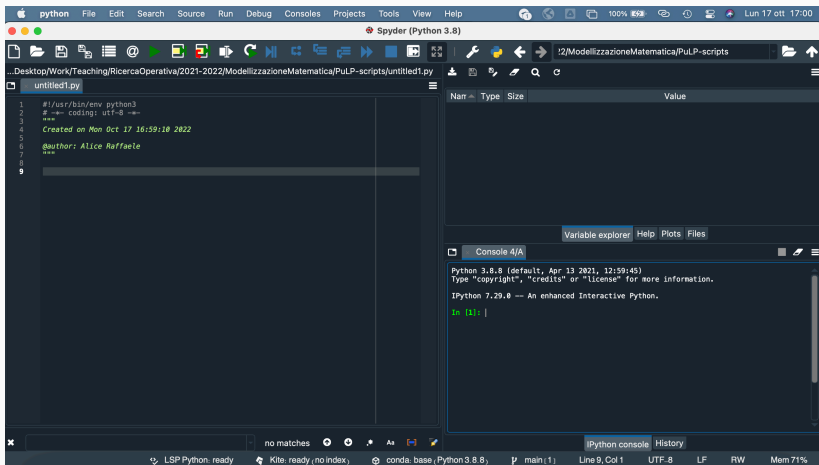
Link: <https://survey.stackoverflow.co/2022/>

- Python è da una decina d'anni uno dei linguaggi più popolari utilizzato dagli sviluppatori.
- Coloro che stanno imparando a programmare lo preferiscono ad altri linguaggi come Java, C++ e C.

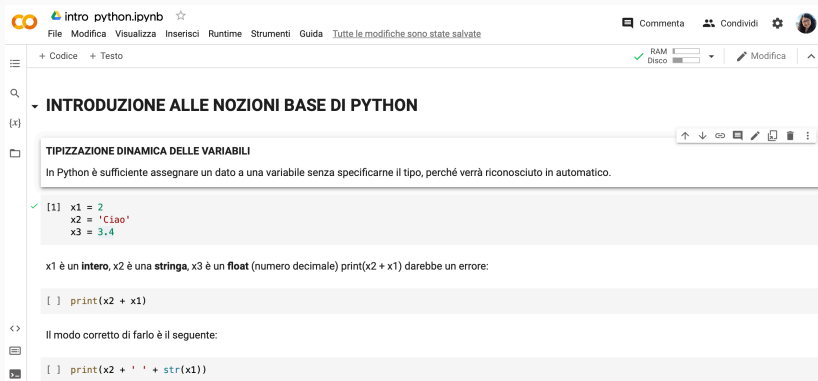
# Come installare e scrivere script in Python sul proprio PC



Link: <https://www.spyder-ide.org>



# Nozioni base di Python con Google Colab



The screenshot shows a Google Colab notebook interface. At the top, the title bar reads 'intro.python.ipynb' with a star icon. Below it, a menu bar includes 'File', 'Modifica', 'Visualizza', 'Inserisci', 'Runtime', 'Strumenti', 'Guida', and a status message 'Tutte le modifiche sono state salvate'. On the right, there are icons for 'Commenta', 'Condividi', and a user profile. A resource monitor shows 'RAM' and 'Disco' usage. The notebook content is organized into sections: a main section 'INTRODUZIONE ALLE NOZIONI BASE DI PYTHON' and a sub-section 'TIPIZZAZIONE DINAMICA DELLE VARIABILI'. The sub-section contains a text block explaining that Python dynamically assigns types to variables. Below this, a code cell shows three lines of code: `x1 = 2`, `x2 = 'Ciao'`, and `x3 = 3.4`. A follow-up text block explains that `x1` is an integer, `x2` is a string, and `x3` is a float, and that attempting to add them would cause an error. A code cell shows an incorrect attempt: `print(x2 + x1)`. Finally, another text block explains the correct way to concatenate, followed by a code cell showing the correct syntax: `print(x2 + ' ' + str(x1))`.

intro.python.ipynb ☆

File Modifica Visualizza Inserisci Runtime Strumenti Guida Tutte le modifiche sono state salvate

+ Codice + Testo

RAM Disco

Modifica

## INTRODUZIONE ALLE NOZIONI BASE DI PYTHON

### TIPIZZAZIONE DINAMICA DELLE VARIABILI

In Python è sufficiente assegnare un dato a una variabile senza specificarne il tipo, perché verrà riconosciuto in automatico.

```
[1] x1 = 2
    x2 = 'Ciao'
    x3 = 3.4
```

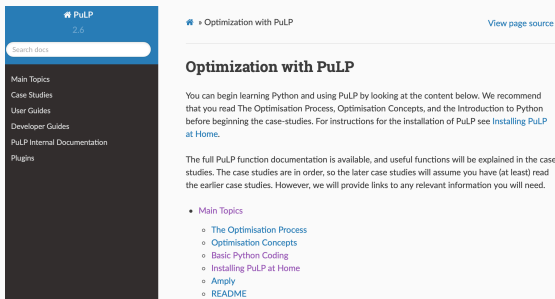
`x1` è un **intero**, `x2` è una **stringa**, `x3` è un **float** (numero decimale) `print(x2 + x1)` darebbe un errore:

```
[ ] print(x2 + x1)
```

Il modo corretto di farlo è il seguente:

```
[ ] print(x2 + ' ' + str(x1))
```

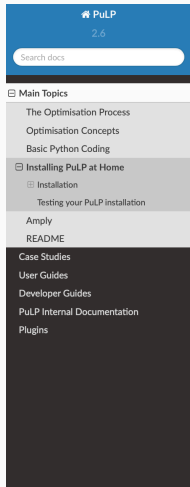
Notebook di Google Colab: <https://shorturl.at/acmZ8>



*<https://coin-or.github.io/pulp/index.html>*

- PuLP è un **LP modeler** scritto in Python.
- Serve a modellizzare e risolvere, tramite un **solver**, problemi di ottimizzazione di Programmazione Lineare, Lineare Intera, e Lineare Mista Intera.

# Come installare PuLP sul proprio PC



## Installing PuLP at Home

PuLP is a free open source software written in Python. It is used to describe optimisation problems as mathematical models. PuLP can then call any of numerous external LP solvers (CBC, GLPK, CPLEX, Gurobi etc) to solve this model and then use python commands to manipulate and display the solution.

### Installation

Note that to install PuLP you must first have a working python installation as described in [installing python](#).

PuLP requires Python  $\geq 2.7$  or Python  $\geq 3.4$ .

The latest version of PuLP can be freely obtained from [github](#).

### Pip and pypi installation

By far the easiest way to install pulp is through the use of [pip](#).

- In windows (please make sure pip is on your path):

```
c:\Python34\Scripts> pip install pulp
```

- In Linux:

```
$ sudo pip install pulp
$ sudo pulptest #needed to get the default solver to work
```

- Then follow the instructions below to test your installation

Tutorial: [https://coin-or.github.io/pulp/main/installing\\_pulp\\_at\\_home.html](https://coin-or.github.io/pulp/main/installing_pulp_at_home.html)

Parte 2:  
Implementare e risolvere  
problemi di Ricerca Operativa  
con Python+PuLP

---



## Riprendiamo un vecchio esercizio...

### “C'è l'insalata!” (cit.)

Un'azienda agricola deve determinare quanti ettari di terreno devono essere dedicati alla produzione di lattuga e pomodori. Si è stimato che, coltivando un ettaro di terreno, si possano produrre annualmente 20 quintali di lattuga e 30 quintali di pomodori. Per portare a termine le coltivazioni, l'azienda dovrà assegnare un suo bracciante a ogni ettaro coltivato a lattuga e due braccianti a ogni ettaro coltivato a pomodori. Per avere sufficiente manodopera per le altre coltivazioni, l'azienda non vuole utilizzare più di 100 lavoratori. Sapendo che l'azienda vende ogni chilogrammo di lattuga e pomodoro rispettivamente a 1 € e a 1.5 €, e vuole assicurarsi un profitto annuo di almeno 50000 € dalla vendita di questi due prodotti, quanti ettari dovrà dedicare alla coltivazione di lattuga e quanti alla coltivazione di pomodori per minimizzare il numero complessivo di ettari coltivati?

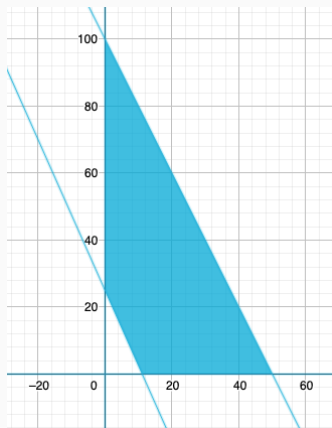
$$\min \quad x_P + x_L$$

$$2x_P + x_L \leq 100$$

$$4500x_P + 2000x_L \geq 50000$$

$$x_P, x_L \geq 0$$

# Risoluzione grafica



Soluzione ottima:

$$x_P = \frac{100}{9} = 11.\bar{1}, x_L = 0, \\ f = \frac{100}{9} = 11.\bar{1}.$$

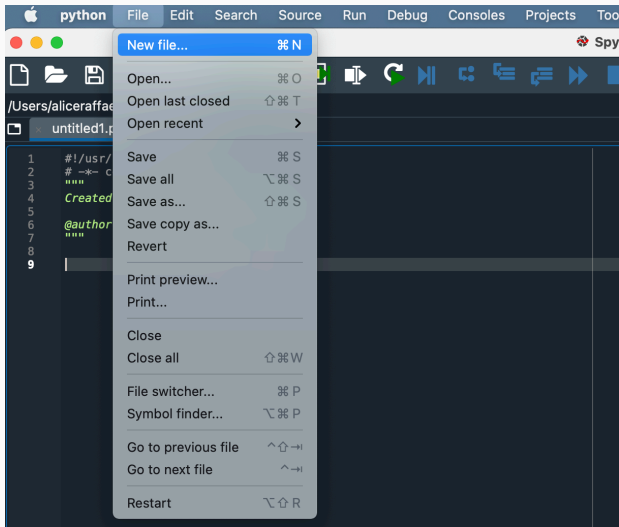
Vertici:

- $(0, 100)$  (Vincolo 1  $\cap x_P = 0$ )  
 $\rightarrow f = 100;$
- $(50, 0)$  (Vincolo 1  $\cap x_L = 0$ )  
 $\rightarrow f = 50;$
- $(25, 0)$  (Vincolo 2  $\cap x_P = 0$ )  
 $\rightarrow f = 25;$
- $(\frac{100}{9}, 0)$  (Vincolo 2  $\cap x_L = 0$ )  
 $\rightarrow f = \frac{100}{9}.$

Come implementare e risolvere questo modello  
matematico tramite Python+PuLP?

# Python+PuLP – 0) Nuovo script in Python

Lanciamo Spyder e creiamo un nuovo script:



## Python+PuLP – 1) Import della libreria PuLP

- PuLP è una libreria di funzioni *esterne* allo script che vogliamo scrivere: è un **modulo**.
- Altri esempi di moduli sono *math*, *numpy*, *pandas*, etc.
- Per poter utilizzare le funzioni esterne dei moduli, è necessario *importarle* scrivendo la seguente istruzione all'inizio dello script:

**import** nome\_libreria

Per *chiamare* una determinata funzione, si scrive:

nome\_libreria.nome\_funzione(parametri)

- Per evitare di scrivere ogni volta *nome\_libreria*, il modulo può anche essere importato nel seguente modo:

**from** nome\_libreria **import** \*

Così, potremo direttamente chiamare la funzione scrivendo:

nome\_funzione(parametri)

## Python+PuLP – 2) Inizializzazione del modello matematico

- La prima funzione di PuLP che sfruttiamo si chiama **LpProblem**:
  - serve a definire e inizializzare un modello matematico di un problema di ottimizzazione;
  - ha due parametri:
    - *name*, una stringa corrispondente al nome che si vuole assegnare al modello;
    - *sense*, ovvero la direzione dell'ottimizzazione, che può essere *LpMinimize* o *LpMaximize*

```
from pulp import *  
model = LpProblem("InsalataPomodori", LpMinimize)
```

LpProblem(name="NoName", sense=const.LpMinimize)  
An LP Problem

- la funzione restituisce poi un oggetto, corrispondente al modello, che viene salvato nella variabile (non decisionale!) chiamata *model* (perché noi abbiamo voluto chiamarla così).

## Python+PuLP – 3) Definizione delle variabili decisionali

- Partendo dalla modellizzazione, procediamo ad aggiungere tutte le componenti del modello.
- Cominciamo dalla definizione delle variabili decisionali, attraverso la funzione **LpVariable**, che ha un parametro obbligatorio e alcuni opzionali:
  - *name* (obbligatorio) stringa corrispondente al nome che si vuole usare nell'output dello script;
  - *lowBound/upBound* (opzionali), il valore minimo/massimo che potrebbe assumere la variabile decisionale;
  - *cat* (opzionale), tipo della variabile decisionale: LpContinuous (default), LpInteger, LpBinary.

```
x_INS = LpVariable()
```

```
LpVariable(name, lowBound=None, upBound=None,  
           cat=const.LpContinuous, e=None)
```

```
This class models an LP Variable with the specified  
associated parameters
```

```
:param name: The name of the variable used in the output .lp  
file :param lowBound: The lower bound on this variable's  
range.
```

```
Default is negative infinity :param upBound: The upper  
bound on this variable's range.
```

```
Default is positive infinity :param cat: The category
```



Per poterle usare nel nostro modello, dobbiamo definire due variabili, continue e non negative, che rappresentino il numero di ettari da coltivare a lattuga e il numero di quelli da coltivare a pomodori.

```
x_INS = LpVariable("Num_ettari_lattuga", lowBound=0, cat=LpContinuous)
x_POM = LpVariable("Num_ettari_pomodori", lowBound=0, cat=LpContinuous)
```

- In questo semplice esercizio, abbiamo solo due vincoli distinti (non della stessa famiglia).
- Non usiamo nessuna funzione in particolare, ma sfruttiamo l'operatore `+=` per aggiungere i vincoli all'oggetto *model*:

`model += espressione_lineare`

Nel caso dei vincoli, l'espressione lineare è un'equazione (`==`) o una disequazione (`<=`/`>=`).

```
model += 1*x_INS + 2*x_POM <= 100
model += 2000*x_INS + 4500*x_POM >= 50000
```

- Per aggiungere la funzione obiettivo, si usa lo stesso operatore:

```
model += espressione_lineare_funzione_obiettivo
```

Tuttavia, in questo caso, l'espressione non è né un'equazione né una disequazione.

```
model += x_INS + x_POM
```

Ora che abbiamo inserito tutte le componenti del modello, possiamo chiamare la funzione **solve**:

```
model.solve()
```

**NB:** non possiamo semplicemente scrivere **solve()** perché la funzione è propria dell'oggetto *model*.

## Python+PuLP – 7) Stampa dei risultati

Dopo l'ottimizzazione, possiamo scoprire il valore delle variabili decisionali e il valore della funzione obiettivo all'ottimo sfruttando un **ciclo for**, l'insieme **model.variables()**, l'attributo **varValue** e **model.objective**:

```
# Valore delle variabili decisionali all'ottimo
for v in model.variables():
    print(v.name, " = ", round(v.varValue,2))

# Valore della funzione obiettivo
print("Numero minimo di ettari richiesti = {}".format(round(value(model.objective),2)))
```

Note:

- **v.name** è proprio il nome della variabile **v** che abbiamo inserito quando abbiamo definito la variabile;
- **round()** è una funzione che riceve come parametri il numero da arrotondare e il numero di cifre decimali da tenere;
- **value(model.objective)** consente di ottenere il valore della funzione obiettivo;
- Possono esserci diversi modi per stampare una stringa

# Implementazione con Python+PuLP

```
from pulp import *

# Inizializzazione del problema assegnando un nome e la direzione dell'ottimizzazione
model = LpProblem("InsalataPomodori", LpMinimize)

# Variabili
x_INS = LpVariable("Num_ettari_lattuga", lowBound=0, cat=LpContinuous)
x_POM = LpVariable("Num_ettari_pomodori", lowBound=0, cat=LpContinuous)

# Vincoli
model += 1*x_INS + 2*x_POM <= 100
model += 2000*x_INS + 4500*x_POM >= 50000

# Funzione obiettivo
model += x_INS + x_POM

# Chiamata al solver
model.solve()

# Stampa soluzione ottima trovata
for v in model.variables():
    print(v.name, " = ", round(v.varValue,2))

# Valore della funzione obiettivo
print("Numero minimo di ettari richiesti = {}".format(round(value(model.objective),2)))
```

# Soluzione con Python+PuLP

```
In [1]: runfile('/Users/aliceraffaele/Desktop/Work/Teaching/RicercaOperativa/
2021-2022/ModellizzazioneMatematica/PuLP-scripts/esempiol_insalata_pomodori.py',
wdir='/Users/aliceraffaele/Desktop/Work/Teaching/RicercaOperativa/2021-2022/
ModellizzazioneMatematica/PuLP-scripts')
Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: Dec 15 2019

command line - /Users/aliceraffaele/opt/anaconda3/lib/python3.8/site-packages/pulp/
apis/./solverdir/cbc/osx/64/cbc /var/folders/g4/2nykfn_n47d6s4xbx9qf74wr0000gn/T/
d9548efa83bd415f82816447e1951638-pulp.mps timeMode elapsed branch printingOptions al
solution /var/folders/g4/2nykfn_n47d6s4xbx9qf74wr0000gn/T/
d9548efa83bd415f82816447e1951638-pulp.sol (default strategy 1)
At line 2 NAME          MODEL
At line 3 ROWS
At line 7 COLUMNS
At line 14 RHS
At line 17 BOUNDS
At line 18 ENDATA
Problem MODEL has 2 rows, 2 columns and 4 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Presolve 2 (0) rows, 2 (0) columns and 4 (0) elements
0  Obj 0 Primal inf 11.111111 (1)
1  Obj 11.111111
Optimal - objective value 11.111111
Optimal objective 11.1111111 - 1 iterations time 0.002
Option for printingOptions changed from normal to all
Total time (CPU seconds):      0.00    (Wallclock seconds):      0.01

Num_ettari_lattuga = 0.0
Num_ettari_pomodori = 11.11
Numero minimo di ettari richiesti = 11.11
```

Parte 3:

Lavoro di gruppo (25 minuti)

---



## Riprendiamo un altro vecchio esercizio...

### In forneria (variante con nuovi dati)

Un fornaio ogni mattina prepara per il suo negozio una gustosa focaccia genovese e una pizza margherita al taglio. Per la preparazione e la cottura di un chilogrammo di pizza, sono richiesti rispettivamente 25 minuti e 15 minuti. Per produrre invece un chilogrammo di focaccia, sono previsti 20 minuti di preparazione dell'impasto e 35 minuti di cottura nel forno. Avendo anche altri prodotti da preparare, il fornaio non può dedicare all'impasto e alla cottura di questi prodotti più di tre e quattro ore, rispettivamente. Inoltre, il fornaio ha già ricevuto una prenotazione di 1.5 Kg di focaccia. Supponendo che il fornaio riesca a vendere tutto ciò che prepara, e sapendo che il prezzo di vendita per focaccia e pizza è rispettivamente 6 euro/Kg e 8 euro/Kg, qual è la produzione giornaliera più redditizia?

$$\max \quad 6x_F + 8x_P$$

$$25x_P + 20x_F \leq 3 \cdot 60$$

$$15x_P + 35x_F \leq 4 \cdot 60$$

$$x_F \geq 1.5$$

$$x_P, x_F \geq 0$$

1. Implementare il modello matematico del problema *In forneria (v2)* con PuLP, scrivendo uno script in Python.
2. Eseguire lo script.

# Correzione – Implementazione con Python+PuLP

```
from pulp import *

# Inizializzazione del problema assegnando un nome e la direzione dell'ottimizzazione
model = LpProblem("InForneria-v2", LpMaximize)

# Variabili
xF = LpVariable("Kg_Focaccia", lowBound=0, cat=LpContinuous)
xP = LpVariable("Kg_Pizza", lowBound=0, cat=LpContinuous)

# Vincoli
model += 25*xF + 20*xP <= 180
model += 15*xF + 35*xP <= 240
model += xF >= 1.5

# Funzione obiettivo
model += 6*xF + 8*xP

# Chiamata al solver
model.solve()

# Stampa soluzione ottima trovata
for v in model.variables():
    print(v.name, " = ", round(v.varValue,2))

# Valore della funzione obiettivo
print("Ricavo max giornaliero per pizza e focaccia = {}".format(round(value(model.objective),2)))
```

# Correzione – Soluzione con Python+PuLP

```
In [1]: runfile('/Users/aliceraffaele/Desktop/Work/Teaching/RicercaOperativa/2021-2022/ModellizzazioneMatematica/PuLP-scripts/forneria.py', wdir='/Users/aliceraffaele/Desktop/Work/Teaching/RicercaOperativa/2021-2022/ModellizzazioneMatematica/PuLP-scripts')
Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: Dec 15 2019

command line - /Users/aliceraffaele/opt/anaconda3/lib/python3.8/site-packages/pulp/apis/./solverdir/cbc/osx/64/cbc /var/folders/g4/2nykfn_n47d6s4xbx9qf74wr0000gn/T/48c25add7edc47d3a1f896e19352cde3-pulp.mps max timeMode elapsed branch printingOptions all solution /var/folders/g4/2nykfn_n47d6s4xbx9qf74wr0000gn/T/48c25add7edc47d3a1f896e19352cde3-pulp.sol (default strategy 1)
At line 2 NAME          MODEL
At line 3 ROWS
At line 8 COLUMNS
At line 16 RHS
At line 20 BOUNDS
At line 21 ENDATA
Problem MODEL has 3 rows, 2 columns and 5 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Presolve 2 (-1) rows, 2 (0) columns and 4 (-1) elements
0  Obj 9 Dual inf 13.999998 (2)
1  Obj 57
Optimal - objective value 57
After Postsolve, objective 57, infeasibilities - dual 0 (0), primal 0 (0)
Optimal objective 57 - 1 iterations time 0.002, Presolve 0.00
Option for printingOptions changed from normal to all
Total time (CPU seconds):      0.00    (Wallclock seconds):      0.01

Kg_Focaccia = 1.5
Kg_Pizza = 6.0
Ricavo max giornaliero per pizza e focaccia = 57.0
```

# Conclusione

---

Implementare e risolvere con Python+PuLP i problemi:

- *BrumBrumBrum* (ROAR I, lezione 2);
- *Parità domestica di genere* (ROAR I, lezione 4).

Consegnare alla Prof.ssa Picchi i due script di Python sviluppati **entro mercoledì 02 novembre**.



*www.menti.com* – Codice: 2273 9669