Corso ITS: *ARTIFICIAL INTELLIGENCE SPECIALIST*

Modulo: **Programmazione Procedurale in Python**

Docente: *Andrea Ribuoli*

Giovedì **23 Gennaio** 2025

09:00 - 14:00

- perché `2 ^ 3 ** 4` è uguale a **83**?
- l'operatore `^` rappresenta l'operazione **bit-wise XOR**
- non stiamo parlando di operatori logici (anche se le regole fondanti sono identiche)
- gli operatori **bit-wise** ripetono l'operazione logica sottostante per tutte le coppie di **bit**

XOR	0	1
0	0	1
1	1	0

- con `^` questa regola fondante opera **su tutti** i bit degli interi coinvolti
- `3 ** 4` , ossia 3 elevato alla quarta, vale **81**
- 81 equivale a: `64 + 16 + 1`
- in notazione binaria `01010001`

128	64	32	16	8	4	2	1
0	**1**	0	**1**	0	0	0	**1**

- 2 in notazione binaria è `00000010`

128	64	32	16	8	4	2	1
0	0	0	0	0	0	**1**	0

- ne risulta che `2 ^ 81` vale:

128	64	32	16	8	4	2	1

128	64	32	16	8	4	2	1
0	**1**	0	**1**	0	0	**1**	**1**

- ossia: `64 + 16 + 2 + 1` e quindi **83**

- gli operatori **bit-wise* sono i seguenti

operazione	simbolo	descrizione
OR	|	logical inclusive OR
AND	&	logical AND
XOR	^	logical exclusive OR

|

OR	0	1
0	0	1
1	1	1

&

AND	0	1
0	0	0
1	0	1

^

XOR	0	1
0	0	1
1	1	0

- `nome = input("Inserisci il tuo nome: ")`
- alla pressione del tasto di **Invio** il valore inserito viene passato alla variabile nome
- dati in ingresso di tipo numerico trattati con **int()** e **float()**

In [9]:
```
## non può essere dimostrato all'interno di Jupyter (serve sessione terminal
```

In [9]:
```
from random import randint
floor = randint(1, 20)
```

```
actualFloor = 0
if floor > 13 :
    actualFloor = floor - 1
else :
    actualFloor = floor
print(actualFloor)
```

```
18
```

- ricordarsi dei due punti `:`
- gli enunciati dopo `if` e dopo `else` possono essere multipli
- è l'indentazione a semplificare la clausola `else` multiriga
- qualora non esistano enunciati per la clausola `else` , questa va **omessa**

In [13]:
```
from random import randint
floor = randint(1, 20)
actualFloor = floor
if floor > 13 :
    actualFloor = floor - 1
print(actualFloor)
```

```
11
```

- l' `if` e l' `else` sono i primi casi di **enunciati composti**
- **intestazione** più **blocco di enunciati**
- in inglese: **header** più **statement block**
- l'intestazione termina con il carattere **due punti**
- un *blocco di enunciati*:
- - inizia nella riga che segue l'intestazione
- - gli enunciati che lo compongono sono **incolonnati a sinistra**
- - gli enunciati che lo compongono sono posizionati **più a destra** rispetto alla intestazione
- - più a destra di **un qualunque numero di spazi**
- Quanto detto non condiziona i **commenti** che si posizionano liberamente

- in **Python** la strutturazione a blocchi del codice è **parte della sintassi**
- in conflitto con le linee guida delle grammatiche **context-free**
- (si pensi alla *minificazione* in JavaScript)
- suggerimenti:
- - evitare duplicazioni nelle diramazioni
- - se opportuno ricorrere alle **espressioni condizionali** `v1 if cond else v2`

In [12]:
```
from random import randint
floor = randint(1, 20)
actualFloor = floor - 1 if floor > 13 else floor
print(actualFloor)
```

```
10
```

- `>` è un **operatore relazionale**
- ha lo stesso significato della notazione matematica
- questo non è vero per tutti gli operatori relazionali (solo per `<` e per `>`)

Python	Matematica	Descrizione
>	**>**	maggiore
>=	**≥**	maggiore o uguale
<	**<**	minore
<=	**≤**	minore o uguale
==	**=**	uguale
!=	**≠**	diverso

- l'**operatore di uguaglianza** (==) è classica fonte di confusione nei neofiti
- viene confuso con il **simbolo di assegnazione** (=)
- esattamente come nel *linguaggio C*
- gli stessi operatori si applicano anche alle **stringhe**

- gli operatori relazionali hanno **precedenza inferiore** a quelli aritmetici
- non ha senso testare la esatta uguaglianza di numeri in virgola mobile (**floating**)
- ordinamento **lessicografico** di stringhe
- ci sono particolarità:
- - le **maiuscole ** precedono le **minuscole**
- - lo spazio ( ) **precede tutti** i caratteri visualizzabili
- le **cifre** precedono le lettere
- l'ordine dei segni di punteggiatura è particolare
- se due stringhe sono uguali fino all'ultimo carattere della più corta, la più lunga è **maggiore**

- enunciati **annidati** (*nested*): un `if` all'interno di un altro `if`

esempio: dal sito della *Agenzia delle Entrate* leggiamo come l'imposta lorda si calcoli applicando al reddito complessivo, al netto degli oneri deducibili, le aliquote per scaglioni... Realizzare un programma che chieda a video il reddito complessivo, al netto degli oneri deducibili, e restituisca l'imposta lorda.

1) Irpef 2025: nuove aliquote

- **aliquota** del 23% per i redditi fino a 28.000 euro,
- **aliquota** del 35% per i redditi superiori a 28.000 euro e fino a 50.000 euro,
- **aliquota** del 43% per i redditi che superano 50.000 euro,

- **eseguire a mano** come test tenendo traccia su carta dei risultati intermedi e finali (*hand-tracing*)

- costrutto **elif**
- diagrammi di flusso (*flowchart*)
- regola: *non far mai entrare una freccia all'interno di una diversa diramazione
- passare da diagramma di flusso a pseudocodice al crescere delle dimensioni
- collaudo: **coverage**
- copertura di tutti i punti di decisione dell'algoritmo implementato
- si predispone un elenco dei casi di prova necessari e dei risultati previsti
- - ogni diramazione abbia un caso di prova
- - inserire un caso di prova per ogni valore limite
- - progettare i casi di prova **prima** di scrivere il codice
- piano di lavoro

- esigenza: valutare una condizione logica in un punto per poi utilizzarla altrove
- risposta: variabili **booleane**
- tipo dato: **bool**
- solamente **due** valori: **True** e **False**
- non sono stringhe !!
- inizializzione (`in_errore = False o in_errore = True`)
- successivamente: `if failed :`
- operatori **booleani**: `and` , `or` , `not`
- considerazioni:
- - confondere *and* e *or* non è infrequente
- - leggibilità: mai confronti diretti con *True* e *False*
- - concatenazione di operatori relazionali (possibile in Python ma ...)
- - valutazione **in cortocircuito** degli operatori *and* e *or*
- - le leggi di *De Morgan* (**not** applicato a espressioni **and** o **or**)