



**MANUEL TROMBETTA E
GAIA PISCOPO**

CYBER SECURITY PER DH

SAST con BANDIT e PYLINT

Start Now →



DEVOPS: UNA MENTALITÀ E UN CAMBIAMENTO CULTURALE



◆ DevOps è un insieme di metodologie, strumenti e pratiche culturali che uniscono sviluppo software (Development) e operazioni IT (Operations). Si tratta di un cambiamento che promuove nuovi modi di lavorare attraverso:

- ✓ Collaborazione 
- ✓ Automazione 
- ✓ Miglioramento continuo 
- ✓ Azione incentrata sul cliente 

COLLABORAZIONE NEL DEVOPS

COLABORAZIONE

- 👉 Team unificati: sviluppo e operazioni lavorano insieme.
- 🔄 Ciclo continuo di feedback e comunicazione costante.
- 🚀 Obiettivo: ridurre silos, migliorare efficienza e velocità di rilascio.



AUTOMAZIONE NEL DEVOPS

AUTOMAZIONE

- ⌚ Eliminazione delle attività manuali nel ciclo di sviluppo.

- 🚀 Maggiore velocità ed efficienza nel rilascio del software.

-  Continuous Integration & Continuous Deployment (CI/CD):

- ✓ Test e rilascio automatizzati.

- ✓ Minori errori, maggiore stabilità.

CI/CD

- ✓ CI (Continuous Integration):

- 📌 Automatizza l'integrazione delle modifiche al codice in un repository condiviso.

- 📌 Test frequenti per garantire la qualità.

- ✓ CD (Continuous Delivery/Deployment):

- 📌 Automatizza testing, integrazione e rilascio in produzione.

- 📌 Permette aggiornamenti frequenti e incrementali.

MIGLIORAMENTO CONTINUO

➡ DevOps si ispira a tre principi chiave:

- ✓ Agile: sviluppo iterativo e adattabile, con feedback continui.
- ✓ Lean: eliminazione degli sprechi per ottimizzare valore e risorse.
- ✓ Kata del miglioramento: pratica continua di piccoli miglioramenti.

🎯 Obiettivi:

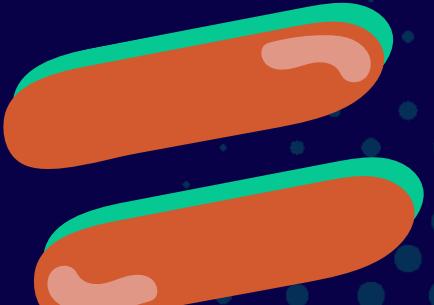
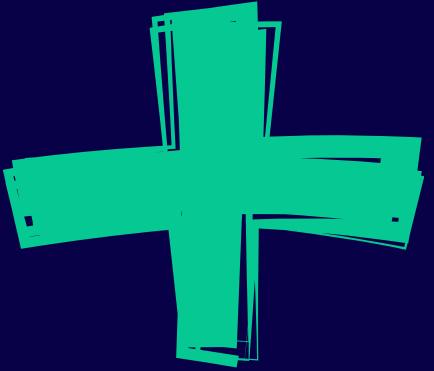
- ✓ Ottimizzare velocità, costi e facilità di rilascio.
- ✓ Supportare la Continuous Delivery per aggiornamenti frequenti.



AZIONE INCENTRATA SUL CLIENTE

- ⌚ Cicli di feedback rapidi
- ✓ Test e rilasci frequenti per adattarsi alle esigenze del mercato.
- ✓ Monitoraggio costante per prevenire problemi.
- ✓ Miglioramento continuo basato sui dati degli utenti.

- 📌 Benefici per il cliente:
 - ✓ Prodotti più affidabili e aggiornati.
 - ✓ Risoluzione veloce dei bug.
 - ✓ Funzionalità migliorate in base alle reali necessità.



SECDEVOPS: EVOLUZIONE DELLA SICUREZZA NEL SOFTWARE

💡 Dall'approccio tradizionale Waterfall a Agile fino a DevOps,
la sicurezza si è evoluta:

Waterfall

- Segue un modello lineare, con la sicurezza spesso relegata alle fasi finali dello sviluppo, rendendo difficile prevenire vulnerabilità nel codice o nelle configurazioni.
- I test di sicurezza sono limitati e frequentemente avvengono dopo i rilasci iniziali.

Agile

- Introduce un approccio iterativo, consentendo modifiche ai requisiti in qualsiasi momento e migliorando la flessibilità del processo di sviluppo.

DevOps

- Integra sviluppo e operations in un unico ciclo di vita, promuovendo efficienza e automazione.
- Riduce drasticamente i tempi di rilascio, passando da mesi a settimane o giorni, adattandosi alle rapide evoluzioni del mercato.

DEVSECOPS VS SECDEVOPS

💡 Entrambi anticipano la sicurezza nel ciclo di sviluppo (shift left):

DevSecOps

- 📌 Integrazione della sicurezza in ogni fase dello sviluppo.
- 📌 La responsabilità è condivisa tra sviluppatori, operations e team di sicurezza.
- 📌 Focus sulla collaborazione tra team per garantire codice sicuro senza rallentare i rilasci.

SecDevOps

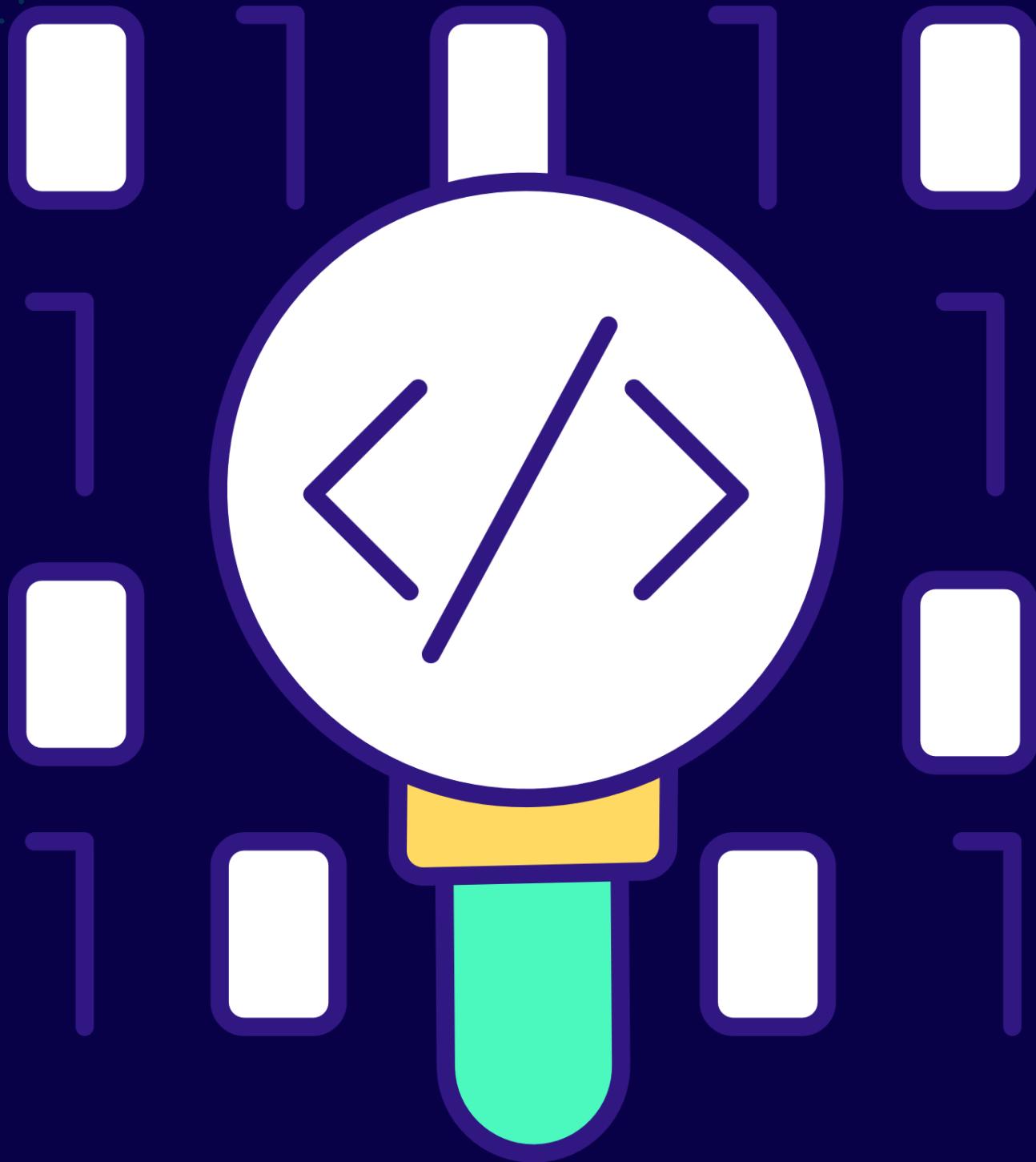
- 📌 La sicurezza è la priorità principale in tutto il ciclo di sviluppo.
- 📌 Gli sviluppatori devono avere competenze avanzate in sicurezza.
- 📌 I test di sicurezza sono continui e obbligatori, riducendo il rischio di vulnerabilità.



TECNOLOGIE DI SICUREZZA NEL SDLC



COS' È IL SAST?



- 🔍 SAST (Static Application Security Testing) è un metodo di white-box testing che analizza il codice sorgente per individuare vulnerabilità di sicurezza.
- 📌 Caratteristiche principali:
 - ✓ Analizza codice sorgente, binario o bytecode.
 - ✓ Non necessita che l'applicazione sia in esecuzione.
 - ✓ Fornisce feedback immediati per correggere le vulnerabilità in tempo reale.
 - ✓ Contribuisce a creare applicazioni più sicure fin dalle prime fasi di sviluppo.

A COSA SERVE IL SAST?

🎯 Obiettivi principali:

- ✓ Individuare falle di sicurezza nelle prime fasi dello sviluppo.
- ✓ Ridurre rischi e costi di correzione delle vulnerabilità.
- ✓ Evidenziare la posizione delle vulnerabilità, semplificando la navigazione del codice.
- ✓ Fornire report dettagliati e dashboard per monitorare e gestire i problemi di sicurezza.



📌 Benefici:

- ◆ Prevenzione proattiva di attacchi.
- ◆ Miglior qualità e affidabilità del software.
- ◆ Maggiore controllo sul ciclo di sviluppo.



COME FUNZIONA IL SAST?

⌚ Passaggi chiave per implementare il SAST:

- ① Scelta dello strumento adatto al linguaggio e framework.
- ② Configurazione: gestione licenze, accessi e risorse.
- ③ Personalizzazione: riduzione dei falsi positivi e aggiornamento delle regole.
- ④ Prioritizzazione delle app critiche e scansioni regolari.

- ⑤ Analisi dei risultati e correzione delle vulnerabilità.
 - ⑥ Formazione del personale per un uso efficace dello strumento.
- 🚀 Obiettivo: garantire sicurezza senza rallentare il ciclo di sviluppo.



LIMITAZIONI E MOTIVAZIONI PER USARE IL SAST

⚠ Limitazioni del SAST

- ✗ Non rileva vulnerabilità esterne (es. API di terze parti).
- ✗ È complementare ad altri strumenti come il DAST, che analizza le applicazioni in esecuzione.

✓ Perché usare il SAST?

- 📌 94% delle applicazioni web contiene bug di sicurezza.
- 📌 Vulnerabilità non risolte possono causare danni finanziari e reputazionali.
- 📌 Integrato nel ciclo di sviluppo software (SDLC), aiuta a costruire applicazioni più sicure e robuste.



PERCHÉ IL SAST È IMPORTANTE?

💡 Il SAST è fondamentale per la sicurezza del software perché:

- ✓ Identifica e corregge vulnerabilità prima del rilascio.
- ✓ Protegge i dati degli utenti, prevenendo danni alla reputazione aziendale.
- ✓ Offre un vantaggio competitivo, garantendo software sicuri e affidabili.

📌 Risultati:

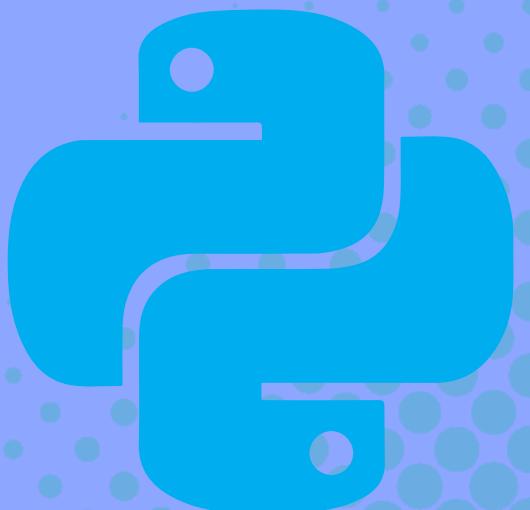
- ◆ Migliore compliance alle normative di sicurezza.
- ◆ Maggiore fiducia da parte degli utenti e delle aziende.
- ◆ Riduzione dei costi e dei tempi di gestione delle vulnerabilità.



LIBRERIA PYTHON BANDIT

 **BANDIT: STRUMENTO
PER INDIVIDUARE
VULNERABILITÀ NEL
CODICE PYTHON.**

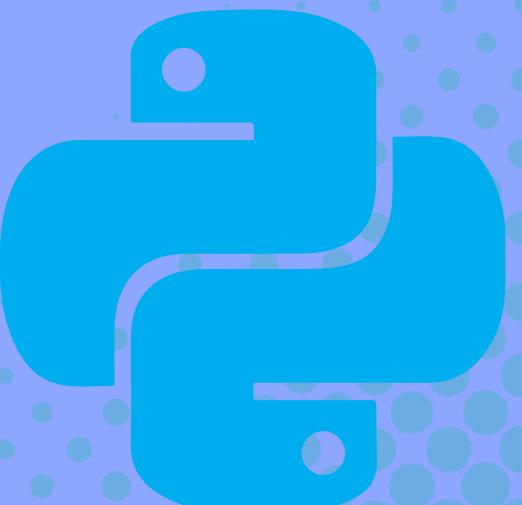
- ❖ Caratteristiche principali:
- ✓ ANALISI RICORSIVA E TEST SPECIFICI.
- ✓ FILTRAGGIO PER LIVELLO DI SEVERITÀ.
- ✓ GENERAZIONE DI REPORT IN VARI FORMATI JSON, CSV, HTML .





ESEMPI DI VULNERABILITÀ IDENTIFICATE DA BANDIT:

- 📌 **B101: USO DELLA FUNZIONE ASSERT POTENZIALMENTE DISABILITABILE IN PRODUZIONE .**
- 📌 **B102: USO DELLA FUNZIONE EXEC PERICOLOSA PER CODICE ARBITRARIO .**
- 📌 **B105: PASSWORD HARDCODED NEL CODICE.**
- 📌 **B301: USO DI PICKLE RISCHIO DI REMOTE CODE EXECUTION .**
- 📌 **B303: USO DI MD5, ALGORITMO CRITTOGRAFICAMENTE DEBOLE.**





COMANDI UTILI CON BANDIT:

💻 ANALIZZARE UNA DIRECTORY INTERA:

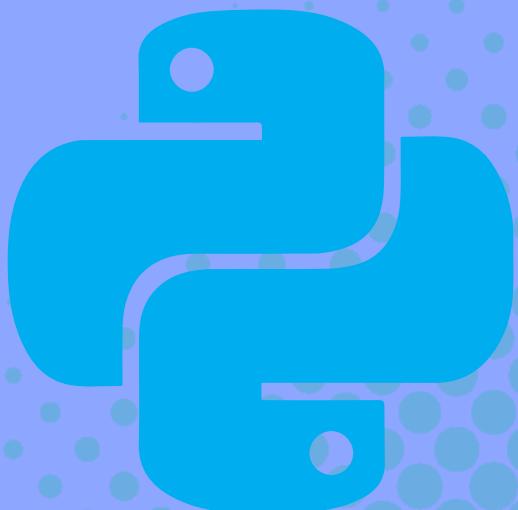
```
bandit -r path/to/your/code
```

🎯 ESEGUIRE TEST SPECIFICI:

```
bandit -t B101,B102 your_script.py
```

📊 GENERARE UN REPORT IN FORMATO JSON:

```
bandit -f json -o report.json your_script.py
```





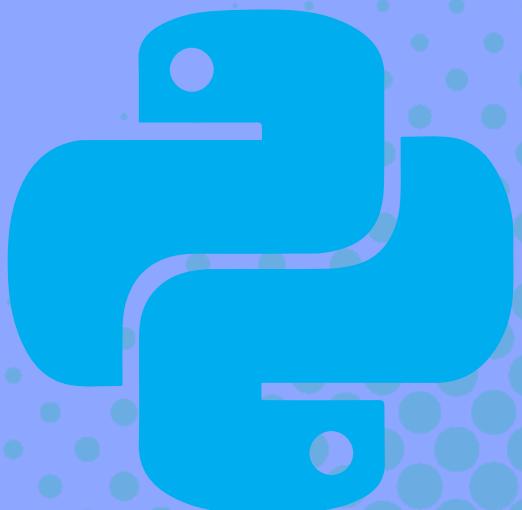
COMANDI UTILI CON BANDIT:

📌 INTEGRAZIONE CON PRE COMMIT:

```
repos:  
  - repo: https://github.com/PyCQA/bandit  
    rev: 'latest'  
hooks:  
  - id: bandit
```

📌 COMMIT & PRE COMMIT

- ◆ **COMMIT:** È UN ISTANTANEA DEL CODICE IN UN DETERMINATO MOMENTO. REGISTRA LE MODIFICHE NEL REPOSITORY GIT, PERMETTENDO DI TRACCIARE L'EVOLUZIONE DEL PROGETTO E RIPRISTINARE VERSIONI PRECEDENTI SE NECESSARIO.
- ◆ **PRE COMMIT:** È UN MECCANISMO CHE ESEGUE CONTROLLI AUTOMATICI PRIMA DI OGNI COMMIT. PREVIENE ERRORI E VULNERABILITÀ BLOCCANDO CODICE NON CONFORME, MIGLIORANDO SICUREZZA E QUALITÀ DEL SOFTWARE.



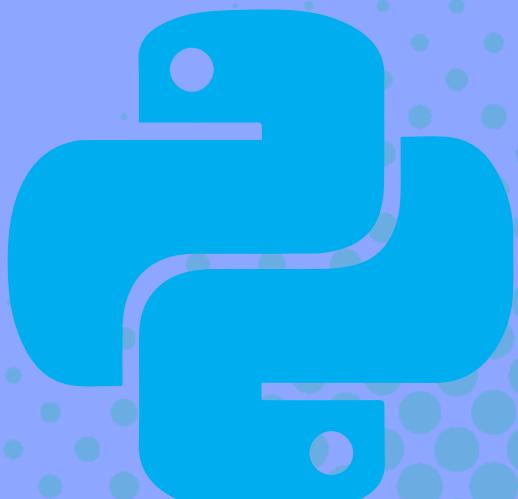
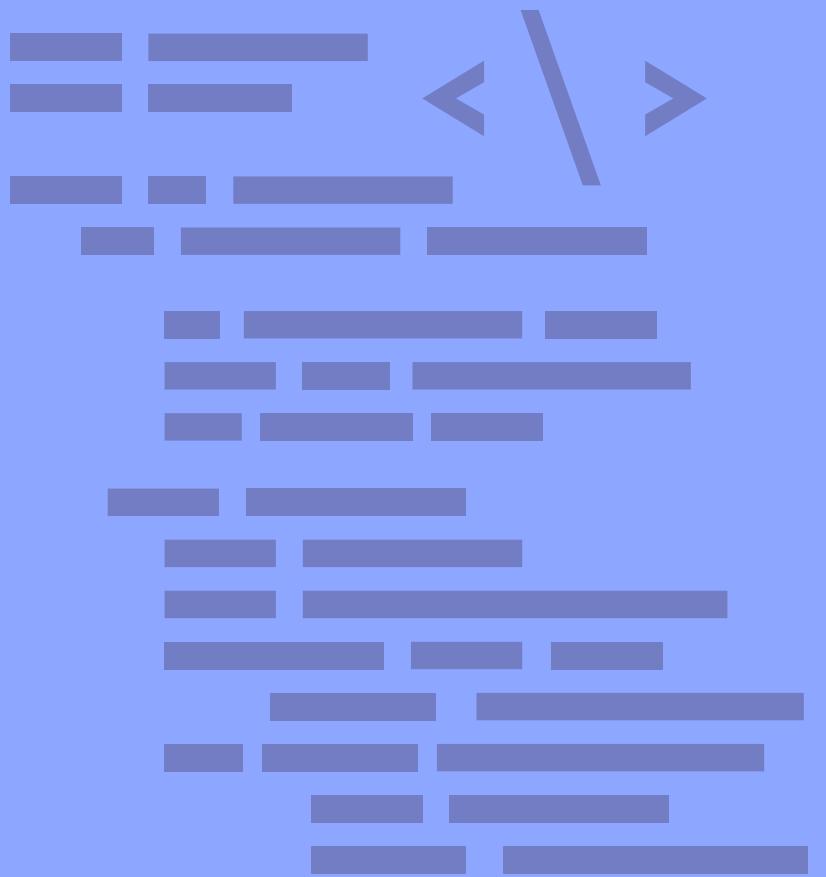


ESEMPIO D' USO DI BANDIT

CODICE TEST:

IL CODICE È UN SEMPLICE PROGRAMMA CHE POTREBBE ESSERE UTILIZZATO NELLE DIGITAL HUMANITIES PER:

- ✓ GESTIRE TESTI DIGITALIZZATI: ELABORAZIONE E MODIFICA DI TESTI.
- ✓ MEMORIZZARE INFORMAZIONI: SALVATAGGIO DI DATI SU FILE.
- ✓ ESEGUIRE OPERAZIONI CRITTOGRAFICHE: HASHING DELLE PASSWORD.
- ✓ CONSENTIRE L' ESECUZIONE DINAMICA DI CODICE MA IN MODO INSICURO! .
- 📌 PROBLEMA: IL CODICE È SCRITTO IN MODO INSICURO, INTRODUCENDO VULNERABILITÀ CRITICHE.



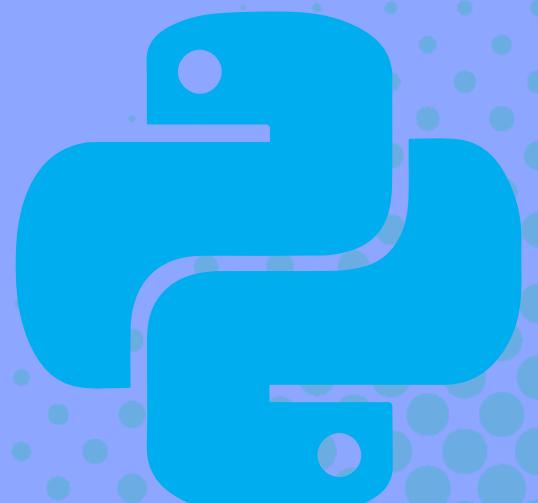
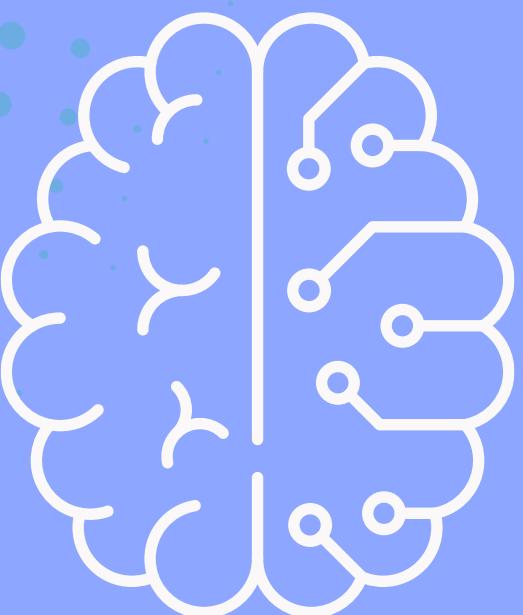


ESEMPIO D' USO DI BANDIT

```
1 import pickle
2 import hashlib
3
4 PASSWORD = "SuperSecret123!"
5
6 def hash_password(password):
7     return hashlib.md5(password.encode()).hexdigest()
8
9 def process_text(text):
10    assert len(text) > 0, "Il testo non può essere vuoto!"
11    local_vars = {}
12    exec("result = text[::-1]", {}, local_vars)
13    return local_vars["result"]
14
15 def save_data(data, filename):
16     with open(filename, 'wb') as f:
17         pickle.dump(data, f)
18
19 def load_data(filename):
20     with open(filename, 'rb') as f:
21         return pickle.load(f)
22
23 user_input = "print('Esecuzione di codice pericoloso!')"
24 exec(user_input)
25
26 if __name__ == "__main__":
27     testo_processato = process_text("Questo è un esempio di Digital Humanities con vulnerabilità.")
28     print("Testo processato:", testo_processato)
29
30     save_data({"username": "admin", "password": PASSWORD}, "dati.pkl")
31     print("Dati caricati:", load_data("dati.pkl"))
32
33     print("Hash della password (MD5, insicuro):", hash_password(PASSWORD))
```

STRUTTURA DEL CODICE:

- IL CODICE È DIVISO IN 5 SEZIONI PRINCIPALI:
- 1 IMPORTAZIONE DELLE LIBRERIE
 - 2 GESTIONE DELLE PASSWORD E HASHING
 - 3 ELABORAZIONE TESTUALE
 - 4 SALVATAGGIO E CARICAMENTO DEI DATI
 - 5 ESECUZIONE DINAMICA DI CODICE





USO DI PICKLE B301 RISCHIO DI REMOTE CODE EXECUTION

CODICE VULNERABILE:

```
import pickle

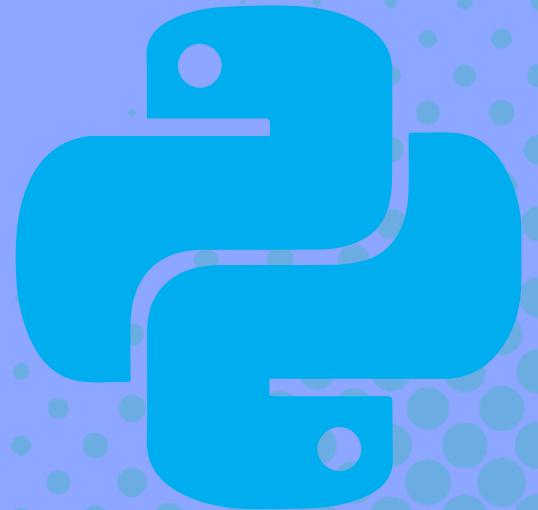
def save_data(data, filename):
    with open(filename, 'wb') as f:
        pickle.dump(data, f)
def load_data(filename):
    with open(filename, 'rb') as f:
        return pickle.load(f)
```

OUTPUT DI BANDIT

Issue: Pickle and modules that wrap it can be unsafe when used to deserialize untrusted data, possible security issue.

Severity: MEDIUM

Confidence: HIGH





PASSWORD HARDCODED B105 DATI SENSIBILI NEL CODICE

CODICE VULNERABILE:

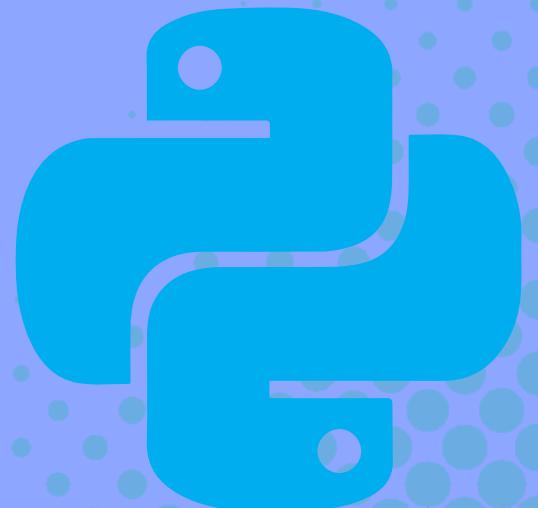
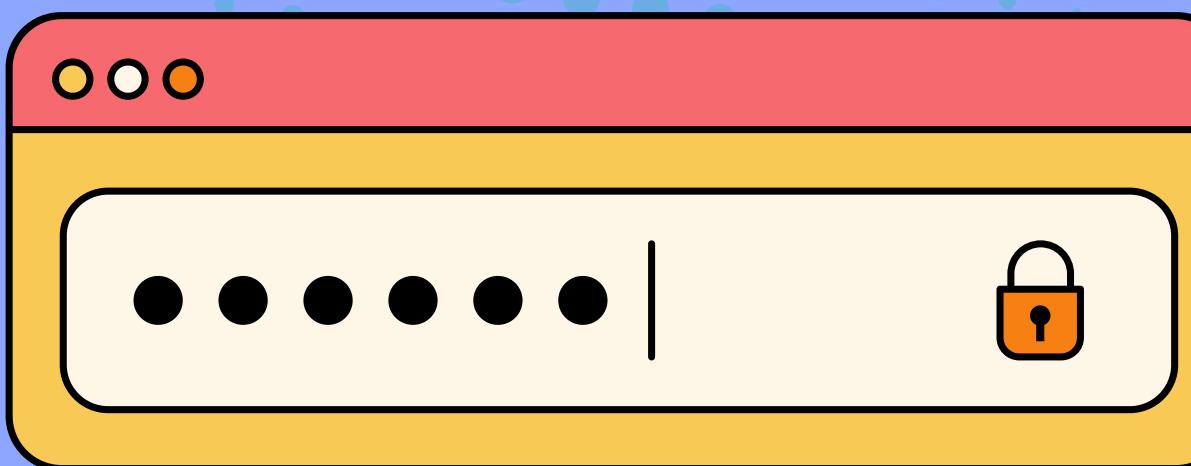
```
# B105: Password hardcoded nel codice
PASSWORD = "SuperSecret123!"
```

OUTPUT DI BANDIT

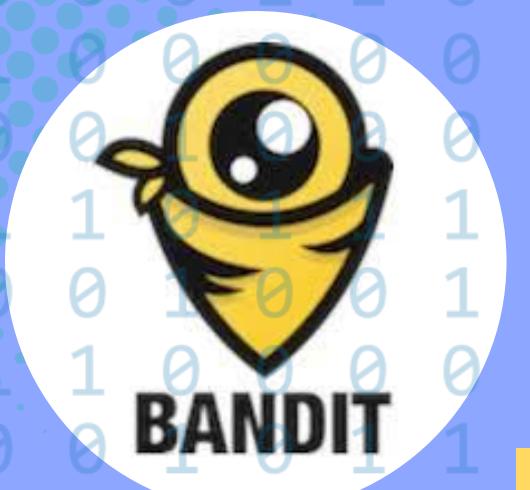
Issue: Possible hardcoded password:
'SuperSecret123!'

Severity: LOW

Confidence: MEDIUM



USO DI MD5 B303 ALGORITMO DI HASHING DEBOLE

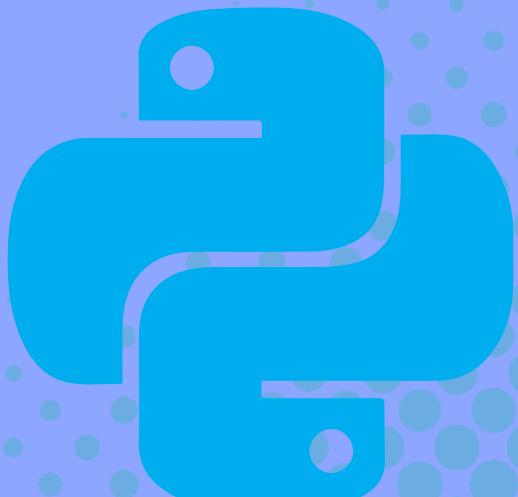


CODICE VULNERABILE:

```
import hashlib # B303: Uso di md5, algoritmo  
crittograficamente debole  
def hash_password(password):  
    return hashlib.md5(password.encode()).hexdigest()  
# B303: Uso di MD5
```

OUTPUT DI BANDIT

```
Issue: Use of weak MD5 hash for security.  
Severity: HIGH  
Confidence: HIGH
```





USO DI ASSERT B101 POSSIBILE RIMOZIONE IN PRODUZIONE

CODICE VULNERABILE:

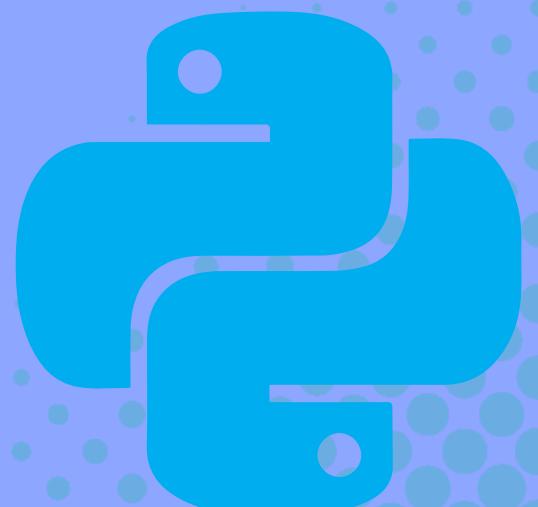
```
def process_text(text):
    assert len(text) > 0, "Il testo non può essere vuoto!"
# B101: Uso di assert
```

OUTPUT DI BANDIT

Issue: Use of assert detected. The enclosed code will be removed when compiling to optimised byte code.

Severity: LOW

Confidence: HIGH





USO DI EXEC B102 RISCHIO DI CODE INJECTION

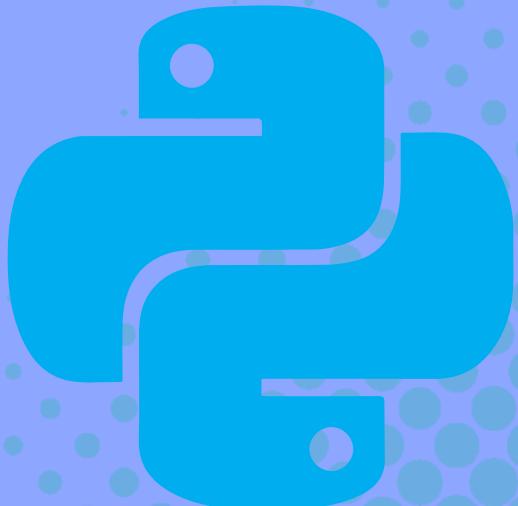
CODICE VULNERABILE:

```
def process_text(text):
    local_vars = {}
    exec("result = text[::-1]", {}, local_vars)
# B102: Uso di exec
    return local_vars["result"]

user_input = "print('Esecuzione di codice pericoloso!')"
exec(user_input) # B102: Uso di exec per eseguire codice arbitrario
```

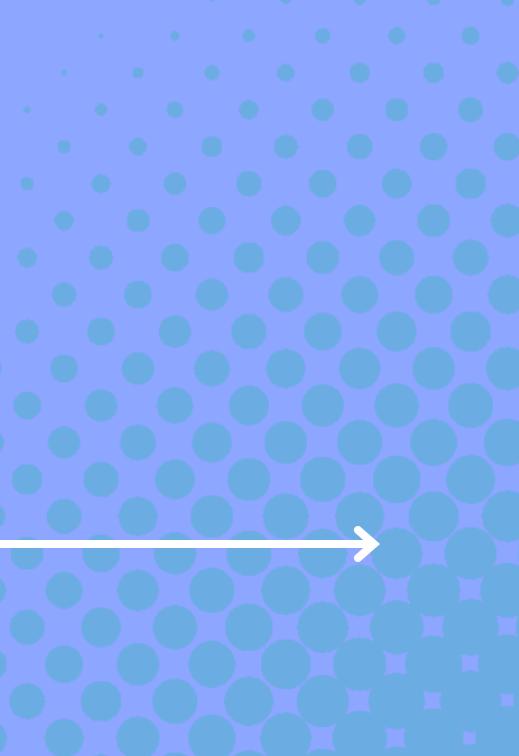
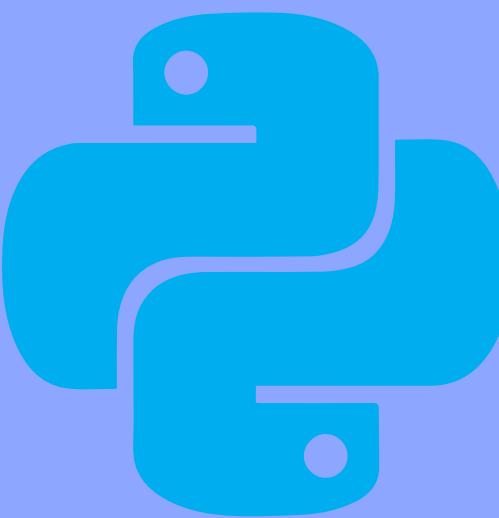
OUTPUT DI BANDIT

```
Issue: Use of exec detected.
Severity: MEDIUM
Confidence: HIGH
```



PROBLEMI DI SICUREZZA DELLE VULNERABILITÀ

Vulnerabilità	Che problema causa?	Rischio principale
Uso di pickle (B301)	pickle può eseguire codice arbitrario quando carica dati, permettendo a un attaccante di infettare il sistema con malware.	Attacco RCE (Remote Code Execution): un hacker può eseguire codice malevolo.
Password hardcoded (B105)	Una password memorizzata direttamente nel codice può essere trovata facilmente (es. tramite una ricerca su GitHub).	Furto di credenziali: chiunque può accedere al sistema usando la password.
Uso di md5 (B303)	MD5 è un algoritmo crittograficamente debole, vulnerabile a attacchi di collisione e brute-force.	Un hacker può decifrare rapidamente gli hash delle password e accedere ai dati.
Uso di assert (B101)	Gli assert possono essere disattivati in produzione (python -O), eliminando le verifiche di sicurezza.	Gli input non vengono controllati correttamente, causando possibili errori o exploit.
Uso di exec() (B102)	exec() esegue codice arbitrario. Se un attaccante fornisce un input malevolo, può prendere il controllo del sistema.	Attacco di Code Injection: l'attaccante può eseguire comandi dannosi sul server.



COME RISOLVERE LE VULNERABILITÀ



Vulnerabilità	Soluzione consigliata	Esempio di codice sicuro
Uso di pickle (B301)	✗ Non usare pickle per salvare dati. ✓ Usare JSON, che è sicuro e non esegue codice.	<pre>import json\nwith open('data.json', 'w') as f:\n json.dump(data, f)</pre>
Password hardcoded (B105)	✗ Non scrivere password nel codice. ✓ Usare variabili d'ambiente o file .env.	<pre>import os\nPASSWORD =\nos.getenv('MY_SECRET_PASSWORD')</pre>
Uso di md5 (B303)	✗ Non usare MD5. ✓ Usare bcrypt o SHA-256, che sono più sicuri.	<pre>import\nhashlib\nhashlib.sha256(password.en\ncode()).hexdigest()</pre>
Uso di assert (B101)	✗ Non usare assert per controlli critici. ✓ Usare un controllo con if e raise Exception.	<pre>if len(text) == 0:\n raise\n ValueError(\"Il testo non può\n essere vuoto!\")</pre>
Uso di exec() (B102)	✗ Evitare exec(), perché permette di eseguire codice malevolo. ✓ Usare funzioni native Python per manipolare dati.	<pre>text[:-1] invece di exec(\"result\n= text[:-1]\")</pre>

IL CODICE FIXATO



BANDIT

```
import json
import hashlib
import os

PASSWORD = os.getenv("MY_SECRET_PASSWORD", "default_password")

def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()

def process_text(text):
    if len(text) == 0:
        raise ValueError("Il testo non può essere vuoto!")
    return text[::-1]

def save_data(data, filename):
    with open(filename, 'w') as f:
        json.dump(data, f)

def load_data(filename):
    with open(filename, 'r') as f:
        return json.load(f)

if __name__ == "__main__":
    testo_processato = process_text("Questo è un esempio di Digital Humanities con sicurezza migliorata.")
    print("Testo processato:", testo_processato)

    save_data({"username": "admin", "password": PASSWORD}, "dati.json")
    print("Dati caricati:", load_data("dati.json"))

    print("Hash della password (SHA-256, sicuro):", hash_password(PASSWORD))
```

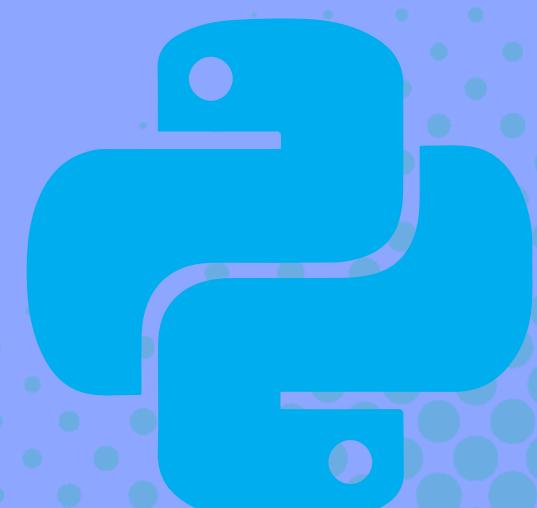
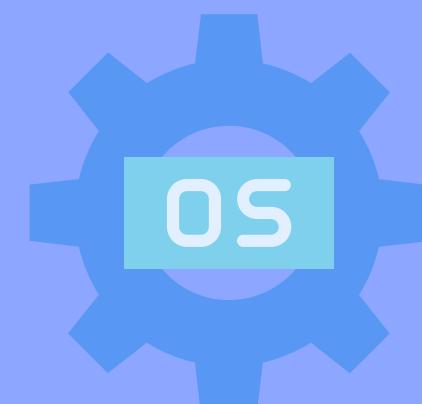


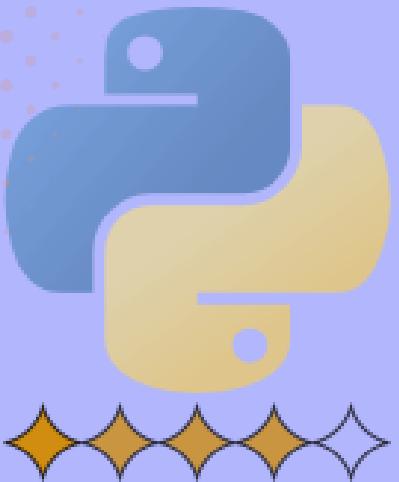
101001011



MIGLIORIE APPORTATE:

- ✓ USO DI JSON INVECE DI PICKLE PER EVITARE ESECUZIONE DI CODICE ARBITRARIO.
- ✓ UTILIZZO DI VARIABILI D' AMBIENTE OS.GETENV. PER EVITARE PASSWORD HARDCODED.
- ✓ SOSTITUZIONE DI MD5 CON SHA 256 PER UN HASHING SICURO.
- ✓ SOSTITUZIONE DI ASSERT CON IF E RAISE EXCEPTION PER EVITARE PROBLEMI IN PRODUZIONE.
- ✓ ELIMINAZIONE DI EXEC , SOSTITUITO CON UNA MANIPOLAZIONE DIRETTA DELLE STRINGHE.



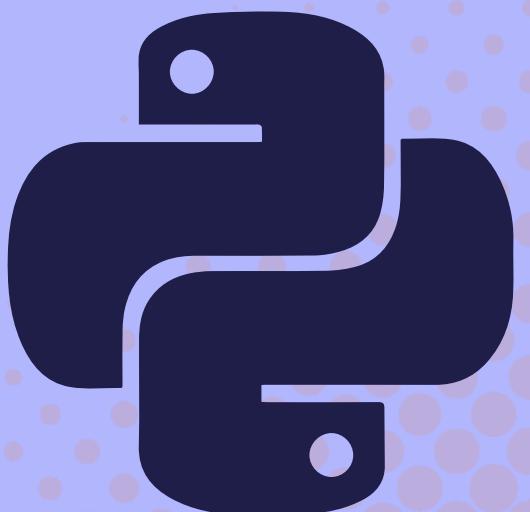


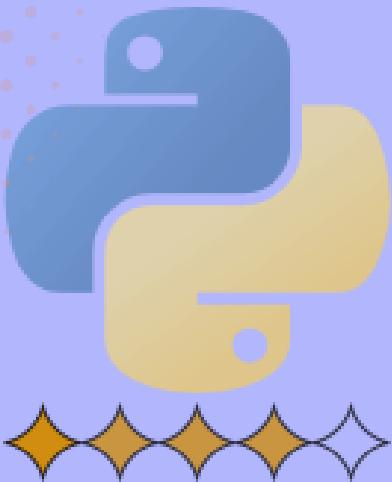
Q PYLINT: STRUMENTO
OPEN-SOURCE PER
L'ANALISI STATICÀ DEL
CODICE PYTHON

PYLINT

⚙ Caratteristiche
principali:

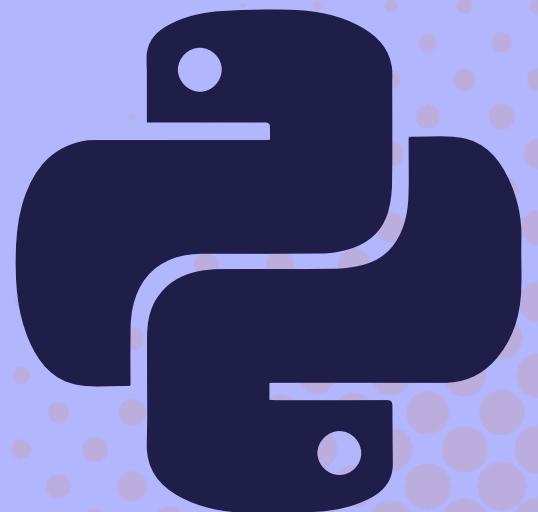
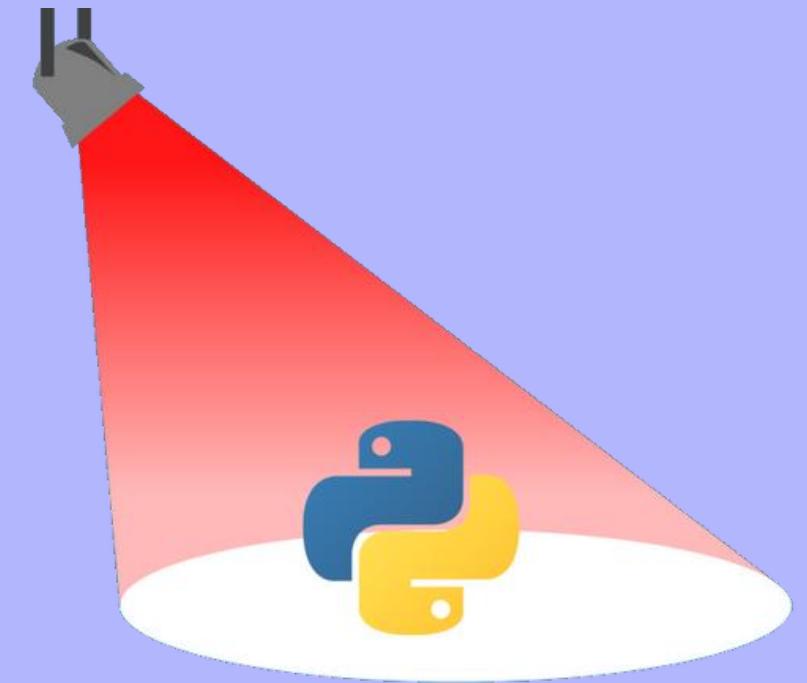
- ✓ IDENTIFICAZIONE DI ERRORI
COMUNI
- ✓ VERIFICA DELLE CONVENZIONI DI
VERIFICA
- ✓ ESTENDIBILITÀ CON PLUGIN DI
SICUREZZA.

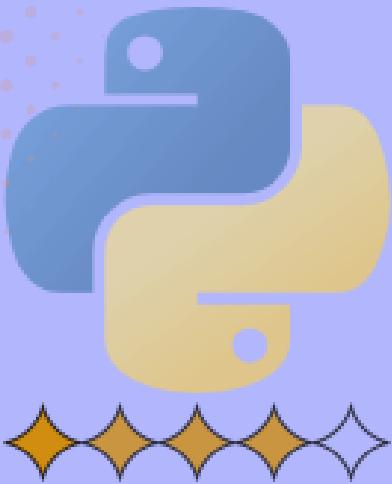




ESEMPI DI VULNERABILITÀ IDENTIFICATE DA BANDIT:

- 📌 **W0311: BAD INDENTATION. FOUND 2 SPACES, EXPECTED 4 BAD INDENTATION .**
- 📌 **C0301: LINE TOO LONG 106 100 LINE TOO LONG.**
- 📌 **R0917: TOO MANY POSITIONAL ARGUMENTS 65 TOO MANY POSITIONAL ARGUMENTS .**
- 📌 **R1705: UNNECESSARY ELSE AFTER RETURN , REMOVE THE ELSE AND DE INDENT THE CODE INSIDE IT NO ELSE RETURN .**
- 📌 **W0621: REDEFINING NAME ANIMALE FROM OUTER SCOPE LINE 224 REDEFINED OUTER NAME .**





COMANDI UTILI CON PYLINT:

ANALIZZARE UNA DIRECTORY INTERA:

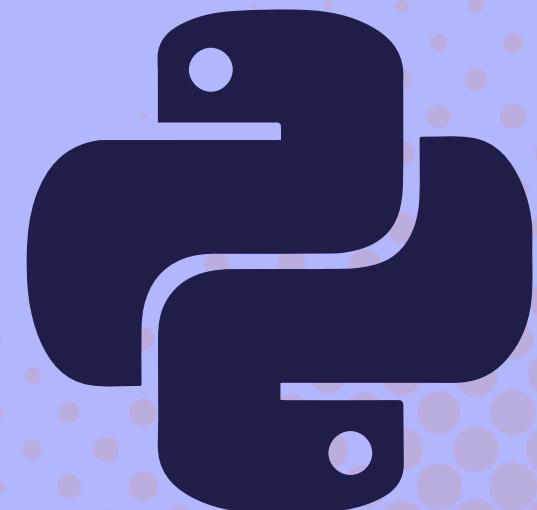
```
python -m pylint path/to/your/code
```

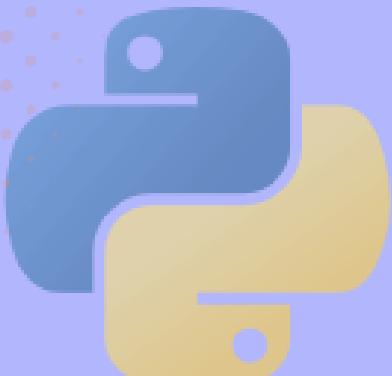
ESEGUIRE TEST SPECIFICI:

```
python -m pylint --disable=all --enable=F  
path\to\your\code
```

GENERARE UN REPORT IN FORMATO JSON:

```
python -m pylint --output-format=json  
path\to\your\code > report.json
```





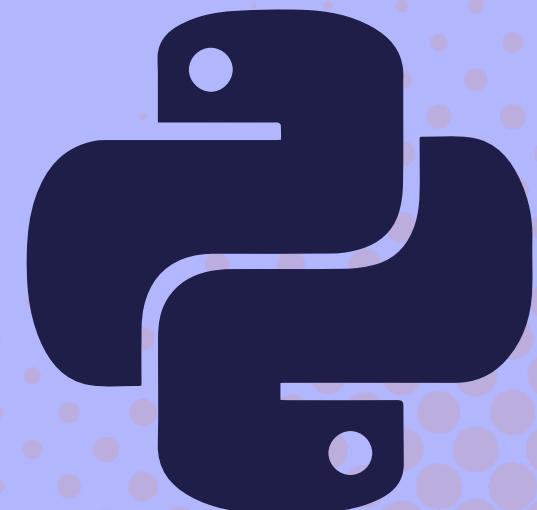
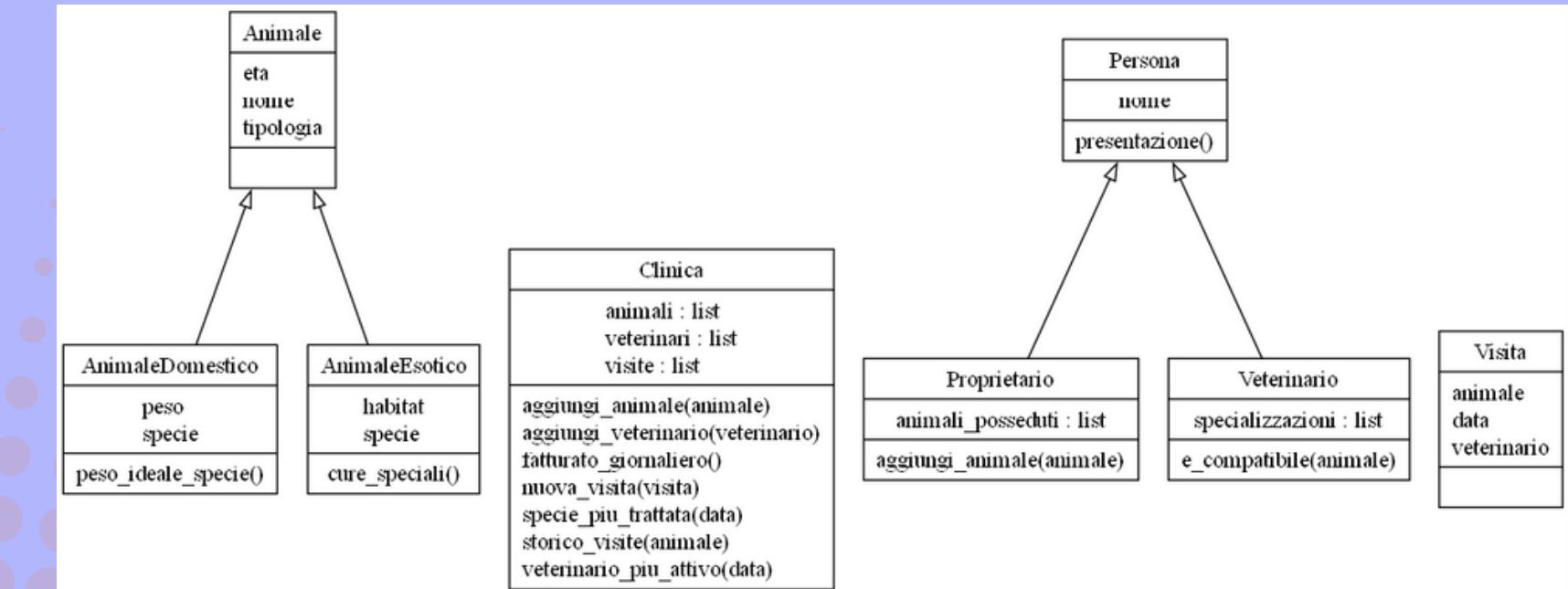
PYREVERSE

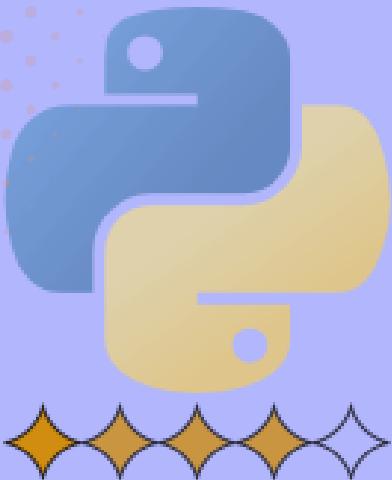
COMANDI UTILI CON PYLINT:

```
pyreverse -o dot -p Esempio  
C:/Users/Uni/Desktop/esempio.py
```

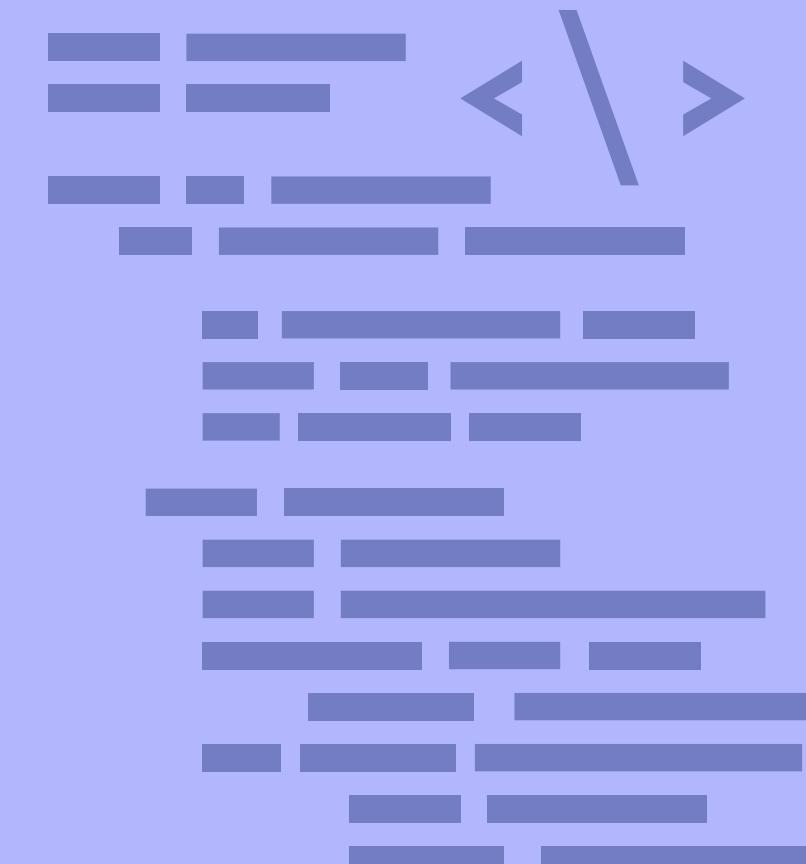
- ◆ ANALIZZA IL CODICE PYTHON E CREA I DIAGRAMMI UML DELLE CLASSI, INCLUSE LE RELAZIONI TRA CLASSI COME EREDITARIETÀ, ASSOCIAZIONI E DIPENDENZE.
- ◆ FUNZIONALITÀ PRINCIPALI DI PYREVERSE

- GENERAZIONE DI DIAGRAMMI DELLE CLASSI: MOSTRA LE CLASSI DEL PROGRAMMA E COME SI COLLEGANO TRA DI LORO EREDITARIETÀ, ASSOCIAZIONI .
- GENERAZIONE DI DIAGRAMMI DELLE DIPENDENZE: VISUALIZZA COME I MODULI DEL PROGETTO SONO CONNESSI.
- SUPPORTO PER L'OUTPUT IN VARI FORMATI: .DOT, .PNG, .SVG





ESEMPIO D' USO DI PYLINT

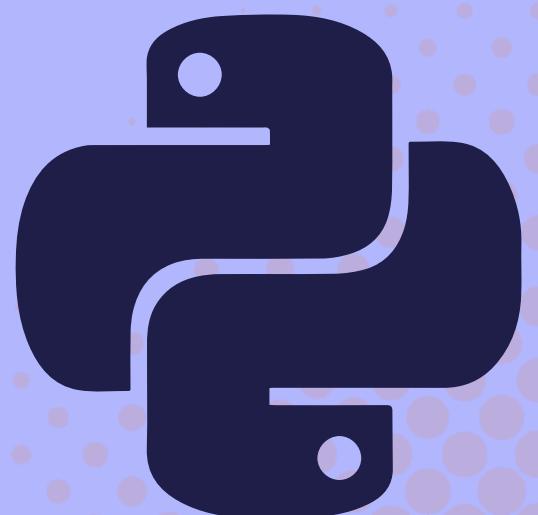


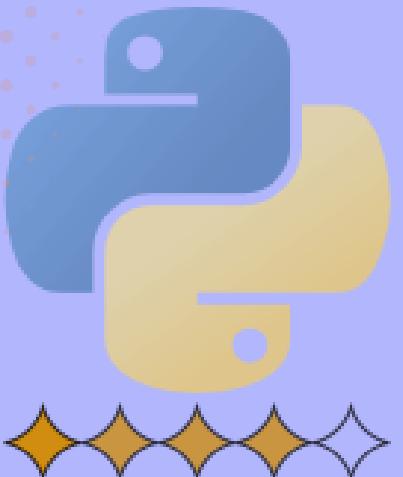
CODICE TEST:

IL CODICE È UN SEMPLICE PROGRAMMA CREATO PER UNA PROVA INTERCORSO UNIVERSITARIA DI INFORMATICA.

- ✓ GESTISCE UNA CLINICA VETERINARIA
- ✓ INCLUDE DIVERSE CLASSI PER RAPPRESENTARE ANIMALI DOMESTICI ED ESOTICI , PERSONE VETERINARI E PROPRIETARI , VISITE DA EFFETTUARE DURANTE GIORNATA ALLA CLINICA.

PROBLEMA: IL CODICE È SCRITTO IN MODO INSICURO, NON NECESSARIAMENTE RISPETTA LE CONVENZIONI DI SCRITTURA.



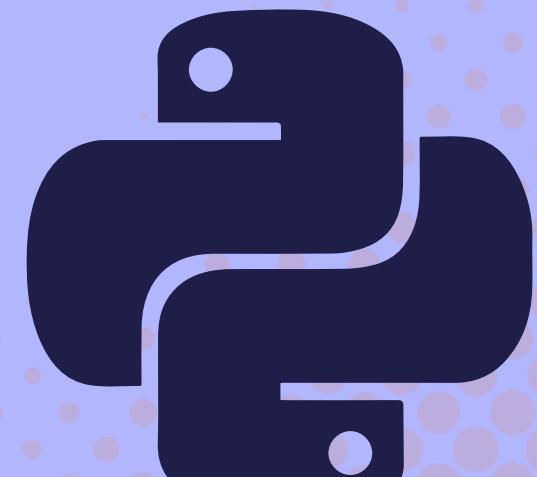


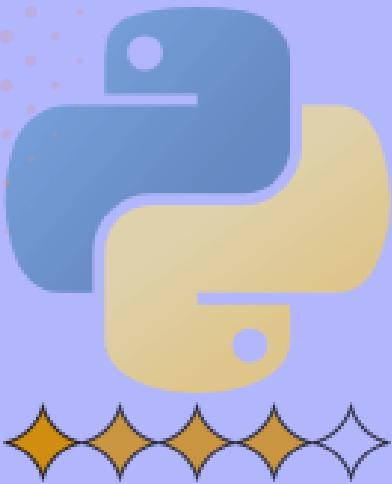
ESEMPIO D' USO DI PYLINT

```
10
11 class Animale:
12     """Classe base per rappresentare un animale."""
13
14     def __init__(self, nome, eta, tipologia):
15         """Inizializza un animale con nome, età e tipologia."""
16         self.nome = nome
17         self.eta = eta
18         self.tipologia = tipologia
19
20
21 class AnimaleDomestico(Animale):
22     """Rappresenta un animale domestico con specie e peso."""
23
24     def __init__(self, nome, eta, tipologia, specie, peso):
25         """Inizializza un animale domestico con specie e peso."""
26         super().__init__(nome, eta, tipologia)
27         self.specie = specie
28         self.peso = peso
29
30
31     def peso_ideale_specie(self):
32         """Determina se il peso dell'animale rientra nell'intervallo ideale."""
33         if self.specie == 'cane' and 30 <= self.peso <= 50:
34             print(f'{self.nome} ha un peso ideale')
35             return True
36         if self.specie == 'gatto' and 10 <= self.peso <= 20:
37             print(f'{self.nome} ha un peso ideale')
38             return True
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
307
308
309
309
310
311
312
313
314
315
316
317
317
318
319
319
320
321
322
323
324
325
326
327
327
328
329
329
330
331
332
333
334
335
336
337
337
338
339
339
340
341
342
343
344
345
346
347
347
348
349
349
350
351
352
353
354
355
356
357
357
358
359
359
360
361
362
363
364
365
366
367
367
368
369
369
370
371
372
373
374
375
376
377
377
378
379
379
380
381
382
383
384
385
386
387
387
388
389
389
390
391
392
393
394
395
396
397
397
398
399
399
400
401
402
403
404
405
406
407
407
408
409
409
410
411
412
413
414
415
416
416
417
418
418
419
419
420
421
422
423
424
425
426
426
427
428
428
429
429
430
431
432
433
434
435
436
436
437
438
438
439
439
440
441
442
443
444
445
446
446
447
448
448
449
449
450
451
452
453
454
455
456
456
457
458
458
459
459
460
461
462
463
464
465
466
466
467
468
468
469
469
470
471
472
473
474
475
476
476
477
478
478
479
479
480
481
482
483
484
485
486
486
487
488
488
489
489
490
491
492
493
494
495
496
496
497
498
498
499
499
500
501
502
503
504
505
506
506
507
508
508
509
509
510
511
512
513
514
515
515
516
517
517
518
518
519
519
520
521
522
523
524
525
525
526
527
527
528
528
529
529
530
531
532
533
534
535
535
536
537
537
538
538
539
539
540
541
542
543
544
545
545
546
547
547
548
548
549
549
550
551
552
553
554
555
555
556
557
557
558
558
559
559
560
561
562
563
564
565
565
566
567
567
568
568
569
569
570
571
572
573
574
575
575
576
577
577
578
578
579
579
580
581
582
583
584
585
585
586
587
587
588
588
589
589
590
591
592
593
594
595
595
596
597
597
598
598
599
599
600
601
602
603
604
604
605
606
606
607
607
608
608
609
609
610
611
612
613
614
614
615
616
616
617
617
618
618
619
619
620
621
622
623
624
624
625
626
626
627
627
628
628
629
629
630
631
632
633
634
634
635
636
636
637
637
638
638
639
639
640
641
642
643
644
644
645
646
646
647
647
648
648
649
649
650
651
652
653
654
654
655
656
656
657
657
658
658
659
659
660
661
662
663
664
664
665
666
666
667
667
668
668
669
669
670
671
672
673
674
674
675
676
676
677
677
678
678
679
679
680
681
682
683
684
684
685
686
686
687
687
688
688
689
689
690
691
692
693
694
694
695
696
696
697
697
698
698
699
699
700
701
702
703
703
704
705
705
706
706
707
707
708
708
709
709
710
711
712
713
713
714
715
715
716
716
717
717
718
718
719
719
720
721
722
723
723
724
725
725
726
726
727
727
728
728
729
729
730
731
732
733
733
734
735
735
736
736
737
737
738
738
739
739
740
741
742
743
743
744
745
745
746
746
747
747
748
748
749
749
750
751
752
753
753
754
755
755
756
756
757
757
758
758
759
759
760
761
762
763
763
764
765
765
766
766
767
767
768
768
769
769
770
771
772
773
773
774
775
775
776
776
777
777
778
778
779
779
780
781
782
783
783
784
785
785
786
786
787
787
788
788
789
789
790
791
792
793
793
794
795
795
796
796
797
797
798
798
799
799
800
801
802
803
803
804
805
805
806
806
807
807
808
808
809
809
810
811
812
813
813
814
815
815
816
816
817
817
818
818
819
819
820
821
822
823
823
824
825
825
826
826
827
827
828
828
829
829
830
831
832
833
833
834
835
835
836
836
837
837
838
838
839
839
840
841
842
843
843
844
845
845
846
846
847
847
848
848
849
849
850
851
852
853
853
854
855
855
856
856
857
857
858
858
859
859
860
861
862
863
863
864
865
865
866
866
867
867
868
868
869
869
870
871
872
873
873
874
875
875
876
876
877
877
878
878
879
879
880
881
882
883
883
884
885
885
886
886
887
887
888
888
889
889
890
891
892
893
893
894
895
895
896
896
897
897
898
898
899
899
900
901
902
903
903
904
905
905
906
906
907
907
908
908
909
909
910
911
912
913
913
914
915
915
916
916
917
917
918
918
919
919
920
921
922
923
923
924
925
925
926
926
927
927
928
928
929
929
930
931
932
933
933
934
935
935
936
936
937
937
938
938
939
939
940
941
942
943
943
944
945
945
946
946
947
947
948
948
949
949
950
951
952
953
953
954
955
955
956
956
957
957
958
958
959
959
960
961
962
963
963
964
965
965
966
966
967
967
968
968
969
969
970
971
972
973
973
974
975
975
976
976
977
977
978
978
979
979
980
981
982
983
983
984
985
985
986
986
987
987
988
988
989
989
990
991
992
993
993
994
995
995
996
996
997
997
998
999
999
1000
1000
```

**STRUTTURA DEL CODICE:
IL CODICE È DIVISO IN 5 SEZIONI
PRINCIPALI:**

- 1 IMPORTAZIONE DELLE LIBRERIE**
- 2 CREAZIONE DELLE CLASSI E SOTTOCLASSI**
- 3 CREAZIONE DI UNA GIORNATA ALLA CLINICA**
- 4 DEFINIZIONE DI ANIMALI, VETERINARI E POSSIBILI VISITE**
- 5 ESECUZIONE DINAMICA DI CODICE**





INVALID NAME (C0103): NAME DOESN'T CONFORM TO NAMING RULES ASSOCIATED TO ITS TYPE (USO SNAKE_CASE)

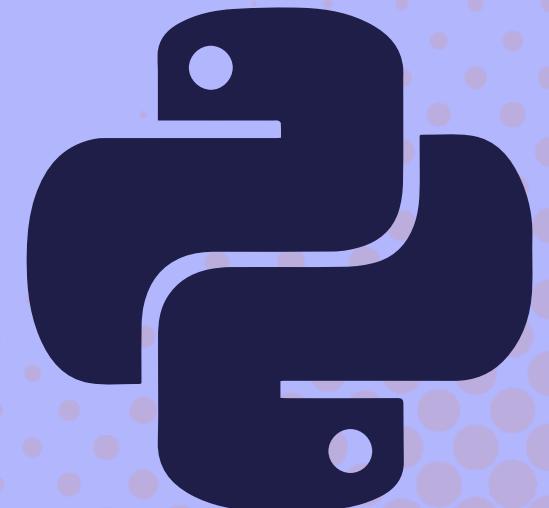
CODICE VULNERABILE:

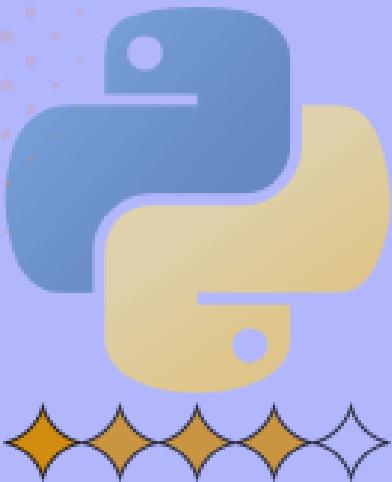
Traccia5ChatGPT.py

OUTPUT DI PYLINT:

```
c:\Users\Uni\Desktop\Traccia5ChatGPT.py :  
1:0: C0103: Module name  
"Traccia5ChatGPT" doesn't conform to  
snake_case naming style (invalid-name)
```

📌 PYLINT SEGUE LE CONVENZIONI DI PEP 8, CHE RACCOMANDA L'USO DELLO SNAKE CASE PER I NOMI DEI FILE PYTHON. QUESTO SIGNIFICA CHE IL NOME DEL FILE DOVREBBE ESSERE SCRITTO TUTTO IN MINUSCOLO, CON LE PAROLE SEPARATE DA UNDERSCORE .





ORDINE DEGLI IMPORT (C0411): MODULI STANDARD VANNO IMPORTATI PRIMA DEI MODULI DI TERZE PARTI

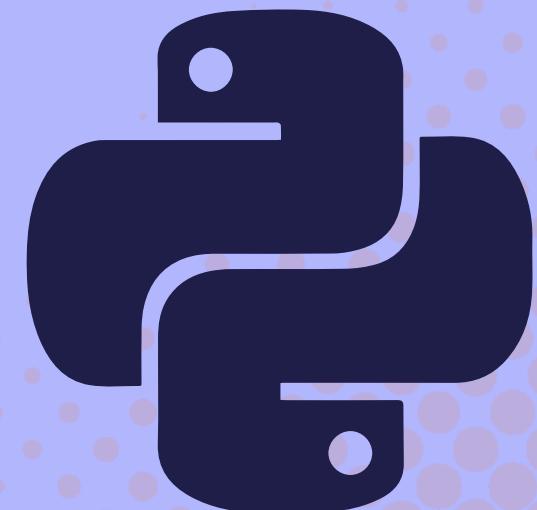
CODICE VULNERABILE:

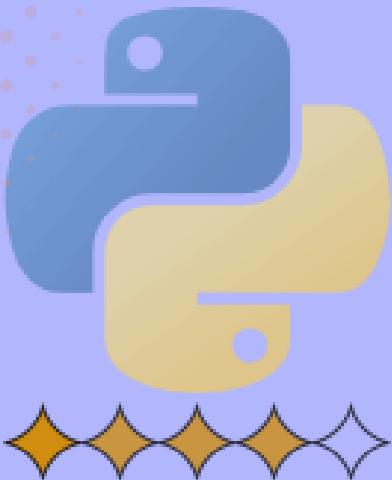
```
from graphviz import Digraph  
import datetime
```

OUTPUT DI PYLINT:

```
c:\Users\Uni\Desktop\Traccia5ChatGPT.py:  
3:0: C0411: standard import "datetime"  
should be placed before third party  
import "graphviz.Digraph" (wrong-import-  
order)
```

📌 ASSICURARSI CHE GLI IMPORT SIANO ORDINATI CORRETTAMENTE. I MODULI STANDARD VANNO IMPORTATI PRIMA DEI MODULI DI TERZE PARTI.





RINOMINAZIONE W0621: REDEFINING NAME VISITA FROM OUTER SCOPE

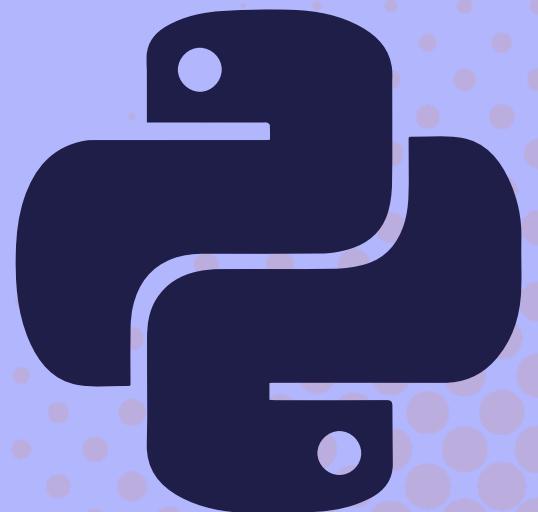
CODICE VULNERABILE:

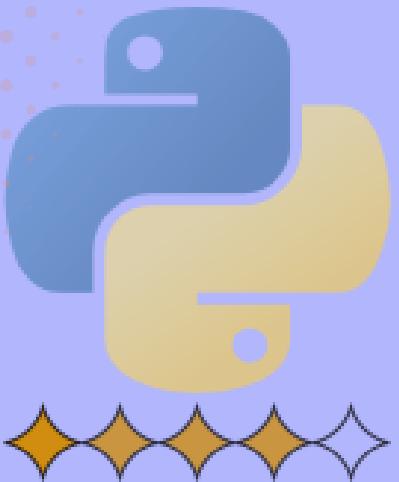
```
for visita in lista_visite:  
    print(f'Visita il {visita.data}  
con il veterinario  
{visita.veterinario.nome}')
```

OUTPUT DI PYLINT:

```
c:\Users\Uni\Desktop\Traccia5ChatGPT.py:  
124:12: W0621: Redefining name 'visita'  
from outer scope (line 205) (redefined-  
outer-name)
```

📌 EVITARE DI USARE LA STESSA VARIABILE VISITA ALL'INTERNO DI LOOP E FUNZIONI, IN MODO CHE NON CI SIA CONFLITTO TRA LE VARIABILI.





IMPORTAZIONE NON UTILIZZATA W0611

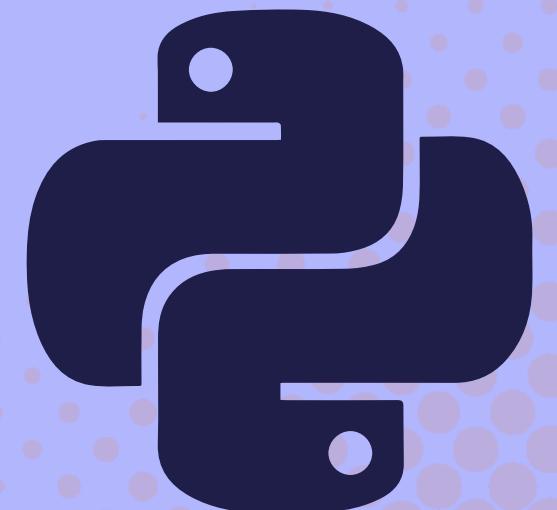
CODICE VULNERABILE:

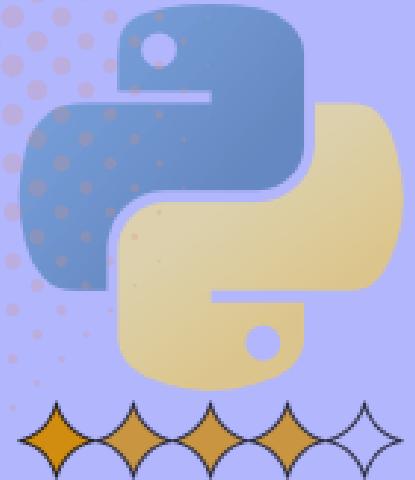
```
import datetime
from graphviz import Digraph
```

OUTPUT DI PYLINT:

```
c:\Users\Uni\Desktop\Traccia5ChatGPT.py:
2:0: W0611: Unused Digraph imported from
graphviz (unused-import)
```

📌 L'IMPORTAZIONE DI DIGRAPH DA GRAPHVIZ NON VIENE MAI UTILIZZATA. ERA STATA EFFETTUATA QUESTA IMPORTAZIONE IN QUANTO ERA STATA SUGGERITA DOPO L'INSTALLAZIONE DI GRAPHVIZ PER LA REALIZZAZIONE DI GRAFICI CON PYREVERSE.





R0913: TOO MANY ARGUMENTS (6/5)

CODICE VULNERABILE:

```
class AnimaleDomestico(Animale):  
    #pylint: disable=too-few-public-methods  
    def __init__(self, nome, eta,  
                 tipologia, specie, peso):  
        super().__init__(nome, eta,  
                         tipologia)  
        self.specie = specie  
        self.peso = peso
```

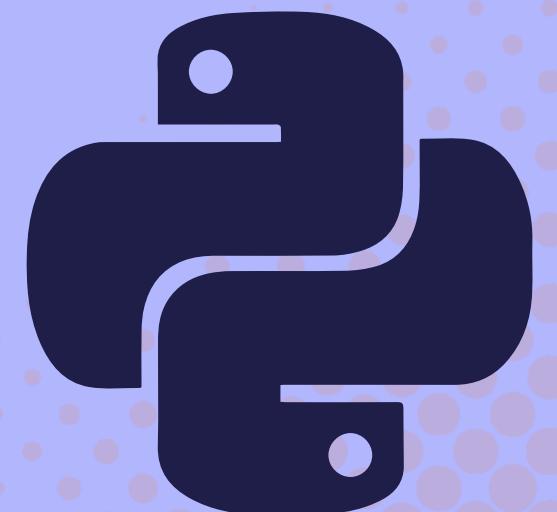
OUTPUT DI PYLINT:

```
c:\Users\Uni\Desktop\Traccia5ChatGPT.py:  
13:4: R0913: Too many arguments (6/5)  
(too-many-arguments)
```

📌 È POSSIBILE CHE VENGANO GENERATI FALSI POSITIVI.

IN QUESTO CASO VIENE FLAGGATO IL RIGO 12 : HA TROPPI ARGOMENTI .

CONSIDERANDO GLI ARGOMENTI DI INIT E SUPER INIT L' UNICO ARGOMENTO MANCANTE CHE RISALTA ALL' OCCHIO È IL SELF CHE PERÒ NON DEVE ESSERE RICHIAMATO.



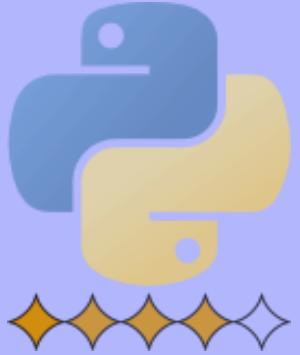
PROBLEMI DELLE VULNERABILITÀ

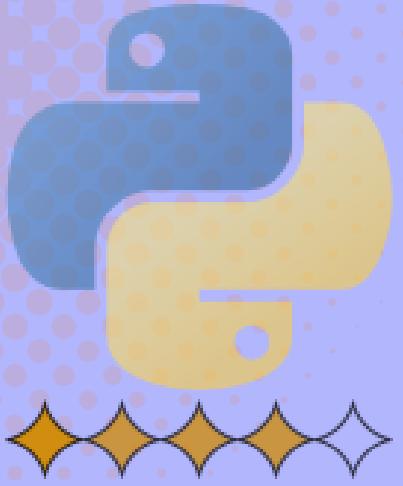


Vulnerabilità	Che problema causa?	Rischio principale
C0103: invalid-name	Il nome della variabile/classe/mетодo non segue le convenzioni (es. usare snake_case per variabili e metodi, CamelCase per classi).	Difficoltà di lettura e mantenibilità del codice.
C0411: Ordine degli import	I moduli standard devono essere importati prima di quelli di terze parti.	Possibili conflitti tra moduli e dipendenze non risolte correttamente.
W0621: rinominazione	Una variabile locale ridefinisce un nome già esistente nello scope esterno.	Può causare bug difficili da individuare e comportamento inatteso.
W0611: Importazione non utilizzata	Un modulo è importato ma mai utilizzato nel codice.	Ingombro inutile, rallentamento della compilazione e codice meno chiaro.
R0913: Too many arguments	Troppi argomenti passati a una funzione/mетодо (6 invece di 5 nel caso segnalato).	Difficile da gestire e testare, possibile violazione del principio di responsabilità singola (SRP).

POSSIBILI SOLUZIONI

Vulnerabilità	Soluzione Consigliata	Esempio di codice corretto
C0103: invalid-name	Seguire le convenzioni di Python: usare snake_case per variabili/metodi e PascalCase per classi.	traccia_5_chat_gpt.py
C0411: Ordine degli import	Importare prima i moduli standard e poi quelli di terze parti.	import datetime from graphviz import Digraph
W0621: rinominazione	Evitare di ridefinire variabili con lo stesso nome già usato nello scope esterno.	for v in visite: clinica.nuova_visita(v)
W0611: Importazione non utilizzata	Eliminare gli import non usati per rendere il codice più pulito.	# Rimuovere questa linea se non la usi from graphviz import Digraph
R0913: Too many arguments	Ridurre il numero di parametri, usando per esempio oggetti dict o dataclass.	[probabilmente è un falso positivo]



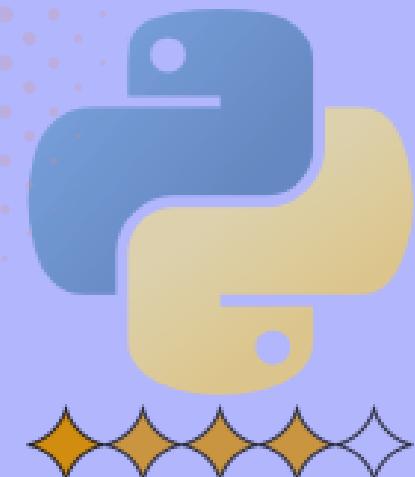


CODICE FIXATO:

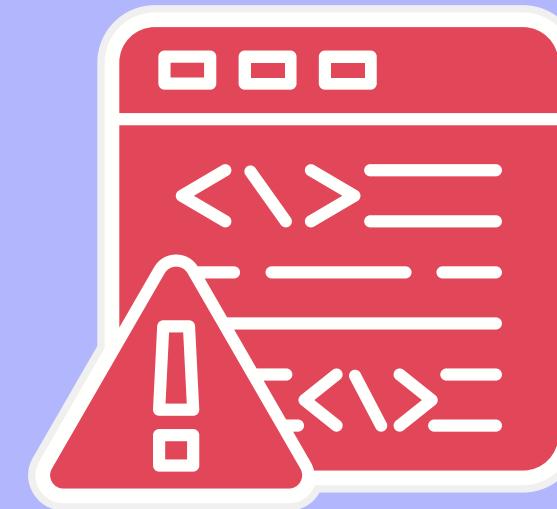
```
#pylint: disable=invalid-name
import datetime
#pylint: disable=trailing-whitespace
class Animale:
    def __init__(self, nome, eta, tipologia):
        self.nome = nome
        self.eta = eta
        self.tipologia = tipologia

class AnimaleDomestico(Animale):
    def __init__(self, nome, eta, tipologia, specie, peso):
        super().__init__(nome, eta, tipologia)
        self.specie = specie
        self.peso = peso
```

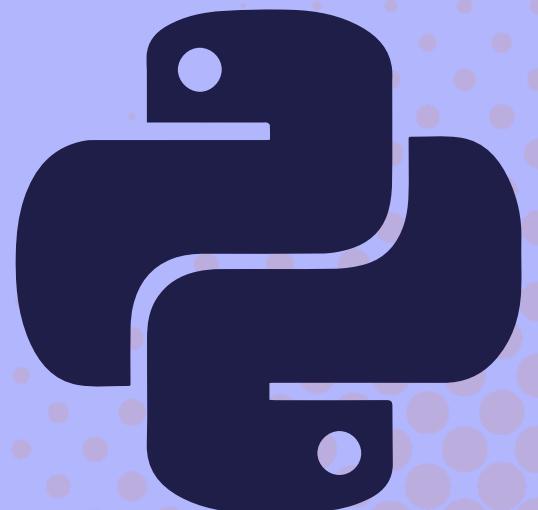
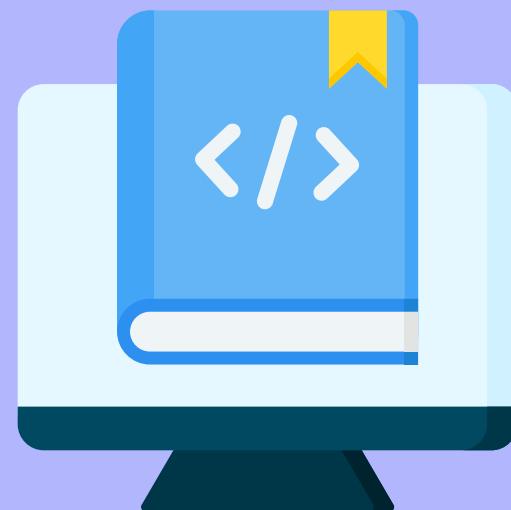
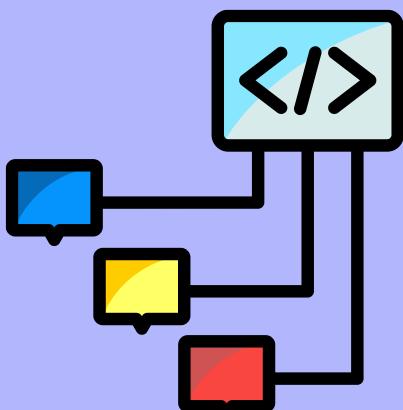
```
#pylint: disable=redefined-outer-name
#Registra una nuova visita se il veterinario è compatibile con l'animale.
def nuova_visita(self, visita):
    if visita.veterinario.e_compatibile(visita.animale): #pylint: disable=trailing-
whitespace
        self.visitate.append(visita)
    print(
        f'Visita registrata per {visita.animale.nome} con {visita.veterinario.nome} il
{visita.data}') # pylint: disable=line-too-long
```



MIGLIORIE EFFETTUATE:

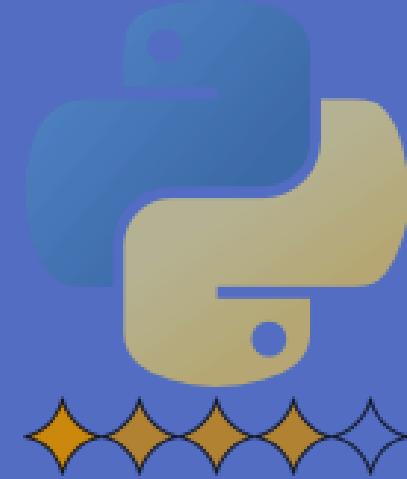


- ✓ USO DEGLI STANDARD DI CODIFICA SECONDO PEP 8, AL FINE DI CONFORMARE LO STILE DEL CODICE.
- ✓ CORREZIONE DELLA SINTASSI USATA NEI PRIMI TENTATIVI DI SCRITTURA DEL CODICE.
- ✓ RIORDINAMENTO DEGLI IMPORT.
- ✓ ELIMINAZIONE DEGLI IMPORT NON UTILIZZATI.
- ✓ MIGLIORE ORGANIZZAZIONE GENERALE DEL CODICE.





BANDIT VS PYLINT



Strumento	Scopo Principale	Focalizzato su	Tipi di Problemi Rilevati
Bandit	Analisi della sicurezza del codice Python	Identificazione di vulnerabilità	<ul style="list-style-type: none">◆ Esecuzione di codice arbitrario (RCE)◆ Injection (SQL, Command)◆ Uso di algoritmi di hashing deboli (es. MD5)◆ Permessi di file non sicuri
Pylint	Analisi della qualità del codice Python	Miglioramento della leggibilità e conformità agli standard di coding	<ul style="list-style-type: none">◆ Errori di sintassi◆ Tipi di dato errati◆ Violazioni delle best practice di PEP8◆ Variabili non utilizzate◆ Nome di funzioni e classi non conformi