Dolnośląski Konkurs INFORMATYCZNY *zDolny Ślązak* dla uczniów szkół podstawowych w roku szkolnym 2025/2026		**ETAP POWIATOWY** **28 listopada 2025 r.** **godz. 12.00** czas trwania 90 minut
Kuratorium Oświaty we Wrocławiu / Dolnośląski Ośrodek Doskonalenia Nauczycieli we Wrocławiu		

Instrukcja przed etapem powiatowym konkursu zDolny Ślązak z informatyki

Na etapie powiatowym zawodnicy rozwiązują test, w którym należy odpowiedzieć na pytania dotyczące programów komputerowych w językach C++ i Python, a odpowiedzi należy zapisać na karcie odpowiedzi, podobnej do karty odpowiedzi z etapu szkolnego. Na rozwiązanie testu zawodnicy mają 90 minut.

Każde zadanie na etapie powiatowym będzie zawierało kod zapisany w C++ i jego równoważny wariant zapisany w Pythonie oraz pewną liczbę pytań dotyczącą analizy tego programu. Pytania będą dobrane w taki sposób, że na niektóre z nich można odpowiedzieć jedynie analizując i rozumiejąc działanie kodu (tzn. bez użycia komputera). Część pytań będzie wymagała umiejętności uruchomienia programów na komputerze, a nawet drobnych modyfikacji zaprezentowanych w arkuszu programów. Wystarczy znajomość jednego z języków programowania.

Podczas rozwiązywania testu zawodnicy mają do dyspozycji komputer z dostępem do internetowego środowiska uruchomieniowego języków C++ oraz Python. W dalszej części tego dokumentu znajduje się instrukcja obsługi środowiska OnlineGDB, które jest zalecanym środowiskiem do pracy i obsługuje oba te języki.

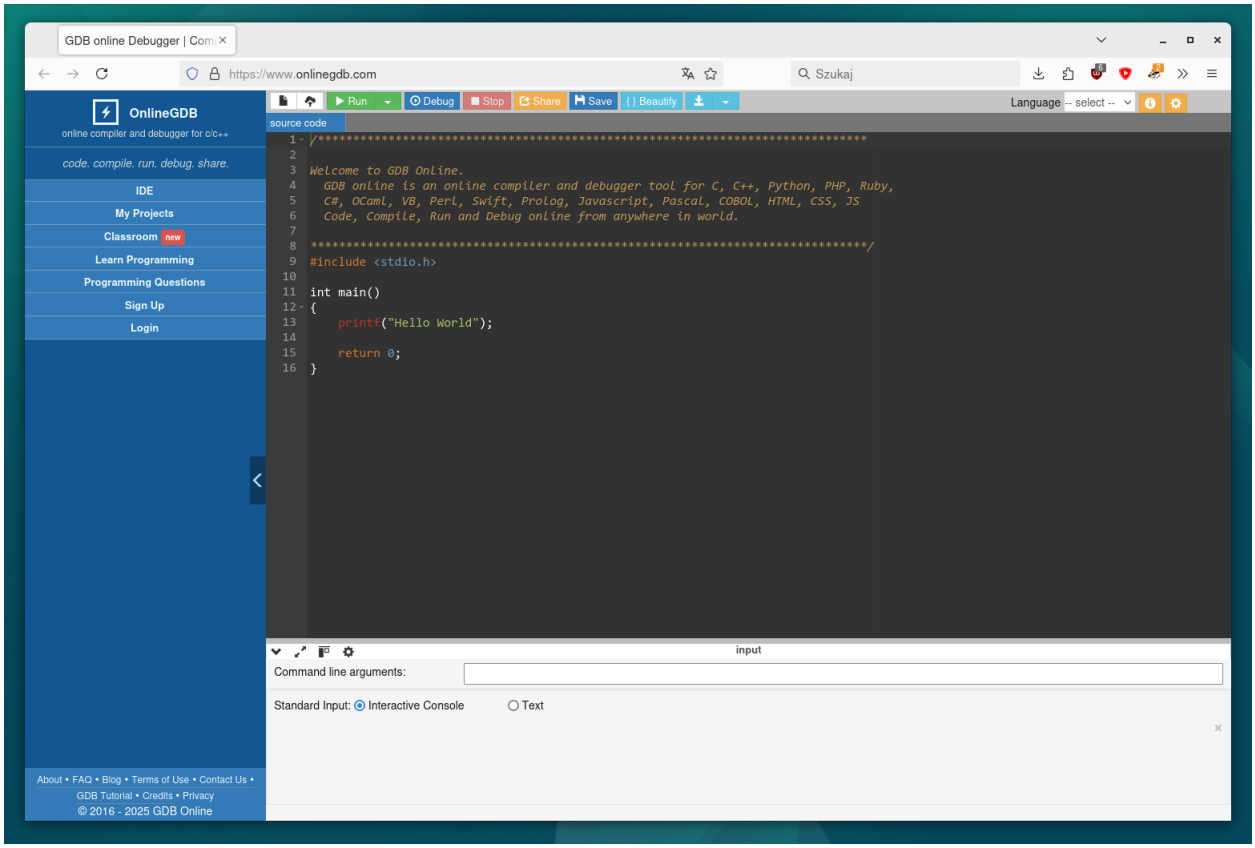
Przykładowe zadanie obrazujące format etapu powiatowego (wraz z jego analizą i przedstawieniem wzorcowego sposobu rozwiązania) znajduje się na końcu tego dokumentu.

Uwaga: Podczas zawodów, zawodnicy mogą używać internetu jedynie w celu pracy w internetowym środowisku uruchomieniowym. **W szczególności zakazane jest komunikowanie się z innymi ludźmi lub używanie narzędzi sztucznej inteligencji.** Członkowie komisji powiatowych czuwają nad poprawnym przebiegiem zawodów i dbają o przestrzeganie przez zawodników powyższej zasady.

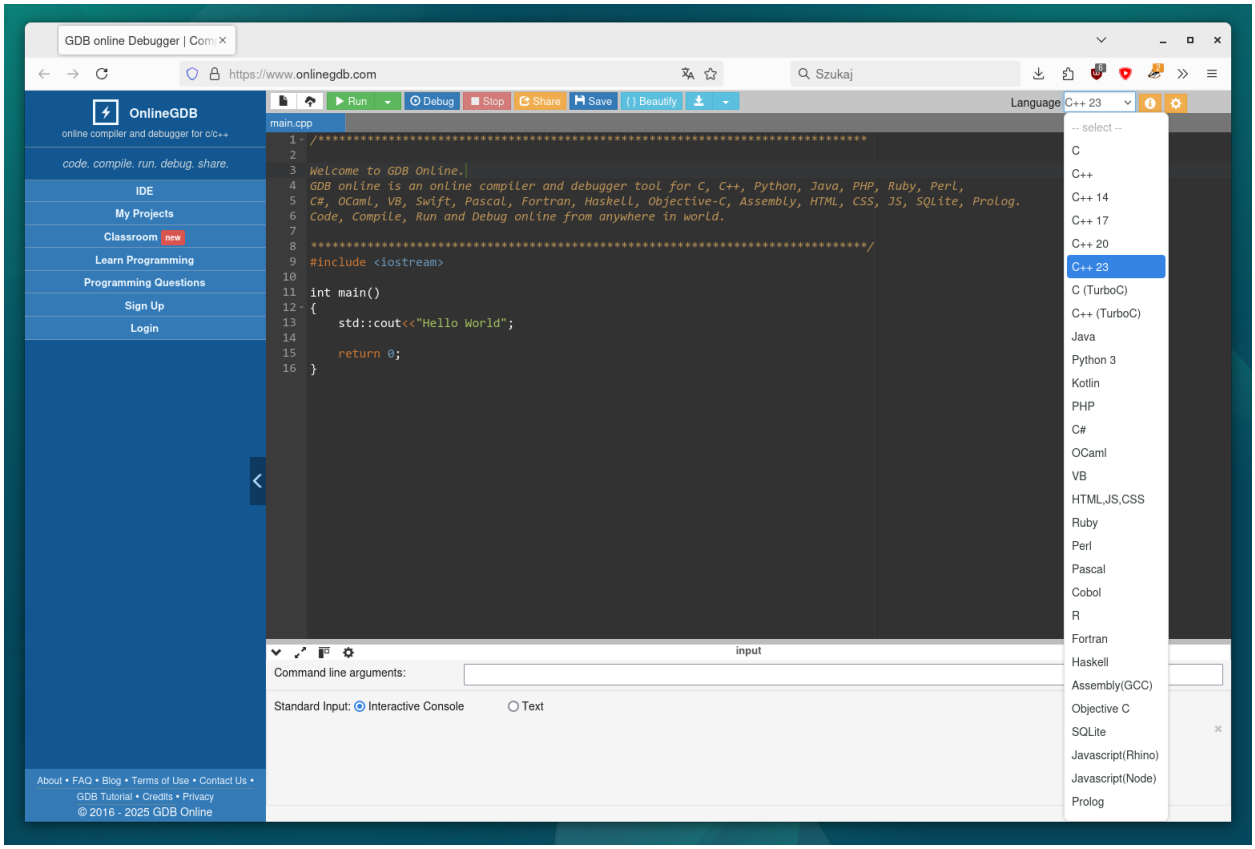
Komisje powiatowe gwarantują działanie środowiska OnlineGDB. Działanie innych (internetowych lub zainstalowanych na komputerach) środowisk nie jest gwarantowane i używanie ich nie jest zalecane, chociaż jest dozwolone (pod warunkiem przestrzegania zasad opisanych powyżej).

Instrukcja obsługi internetowego środowiska uruchomieniowego OnlineGDB

W przeglądarce internetowej wprowadzamy adres strony `https://onlinegdb.com`. Naszym oczom powinna ukazać się następująca strona:



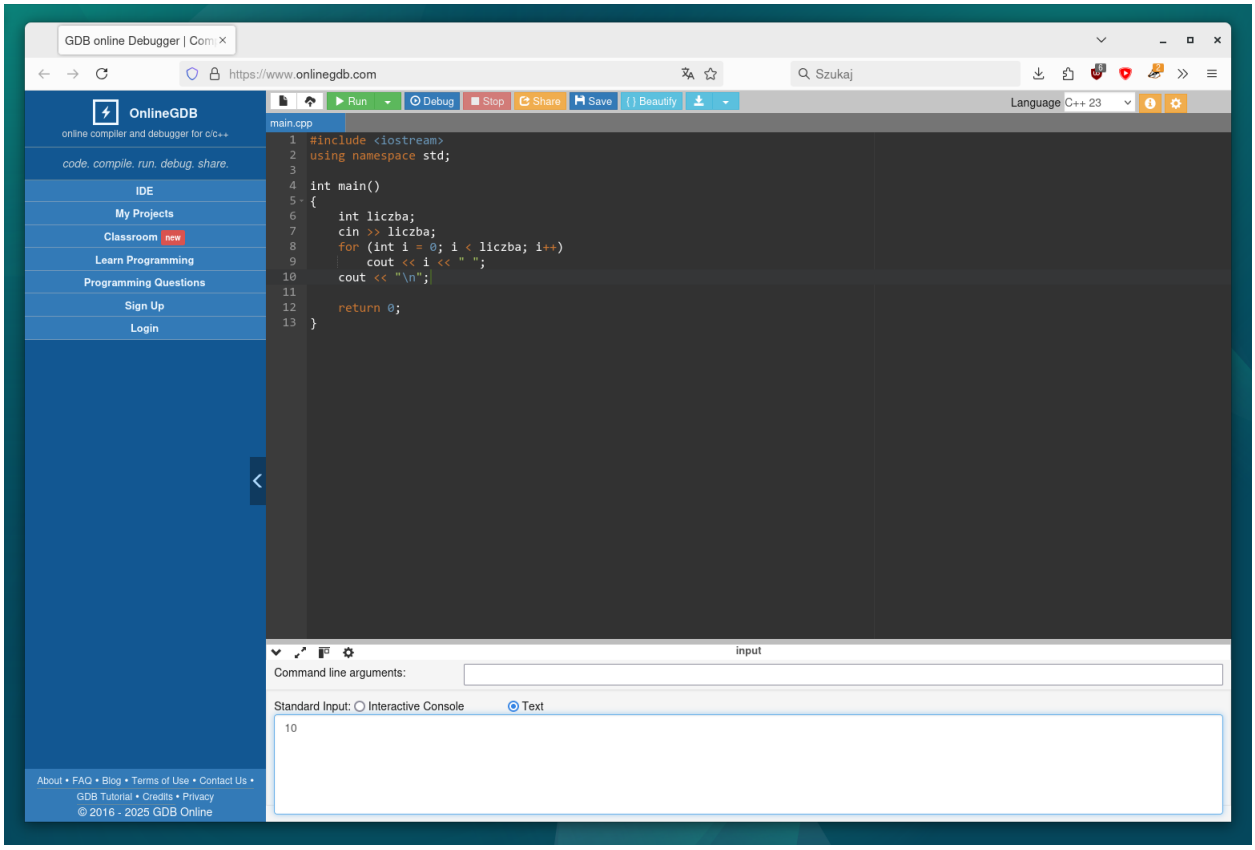
Pierwszym krokiem powinno być wybranie języka programowania z rozwijalnej listy w prawym górnym rogu strony obok napisu *Language*. Zalecamy wybór **C++23** lub **Python 3**.



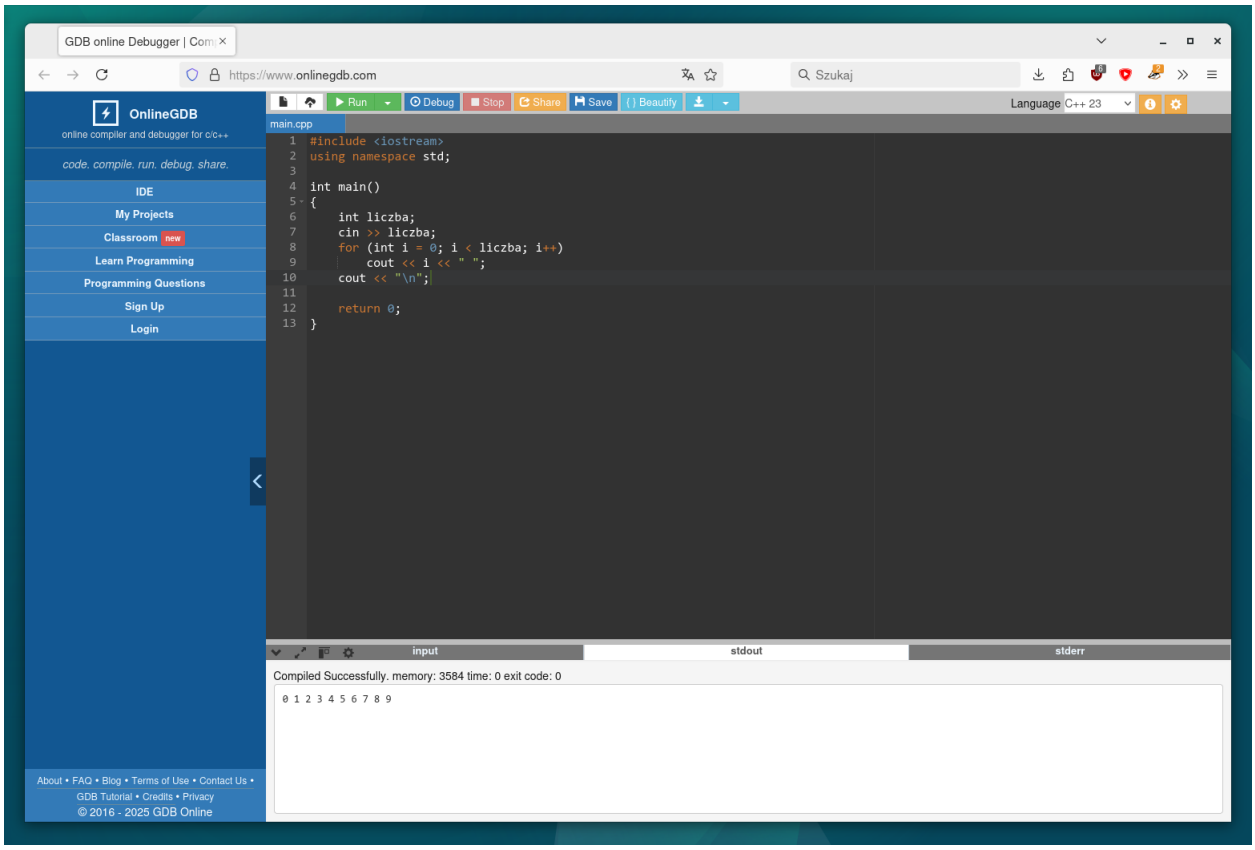
Po wybraniu języka, strona uzupełni kod (na czarnym tle) stosownym przykładem. W tym miejscu należy wpisać kod programu. Można pisać własny program lub przepisać z kartki program z zadania. Możliwe jest zmienianie lub wzbogacanie programu z kartki o swoje własne instrukcje.

Po napisaniu programu, w dolnej części okna możemy wybrać opcję *Text* obok *Standard Input*. W polu poniżej możemy wtedy wprowadzić dane, które chcemy przekazać na wejściu do programu.

W poniższym przykładzie napisaliśmy program, który wczytuje z wejścia liczbę naturalną i wypisuje na wyjście ciąg nieujemnych liczb całkowitych mniejszych od liczby wczytanej z wejścia. Postanawiamy go uruchomić dla danej 10.

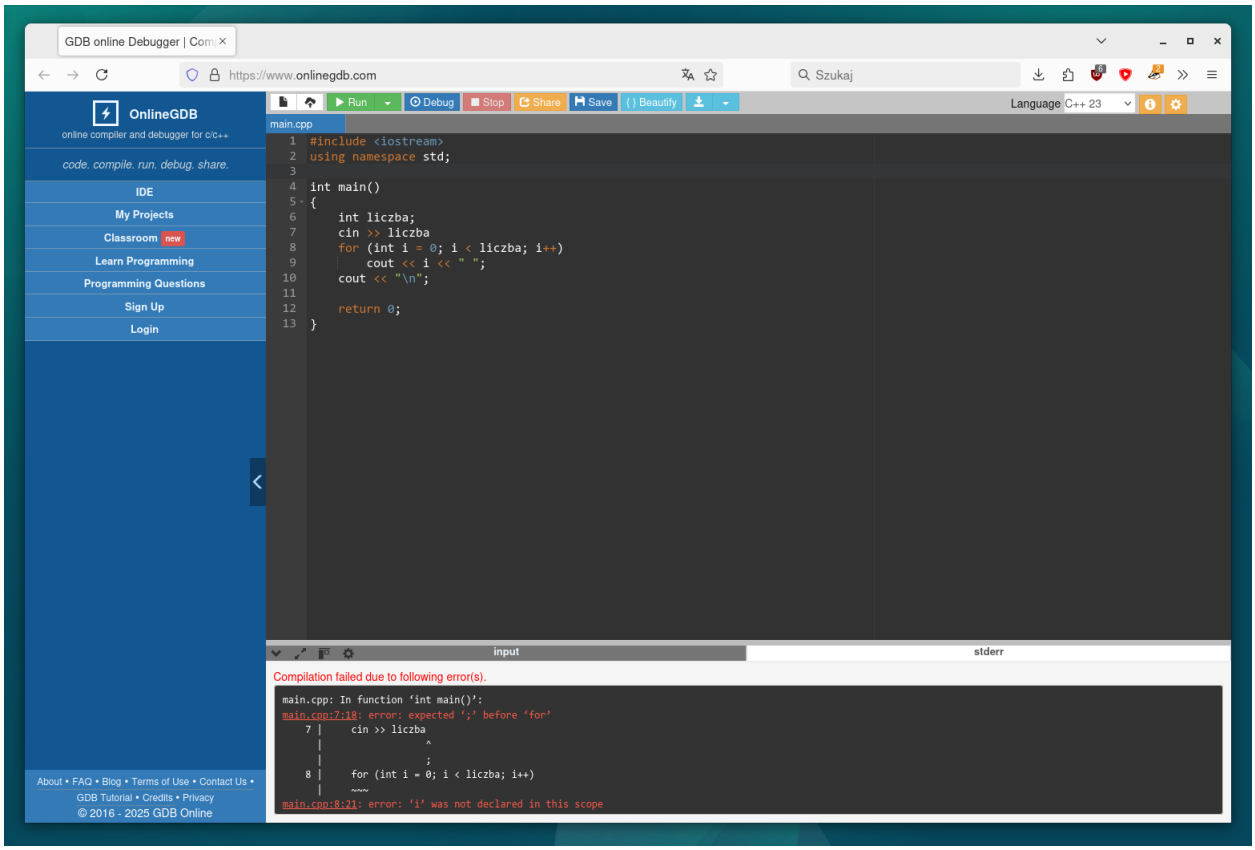


W górnej części okna znajduje się przycisk *Run*, który pozwala uruchomić program. Po jego kliknięciu, w dolnej części okna otrzymamy wynik działania programu.

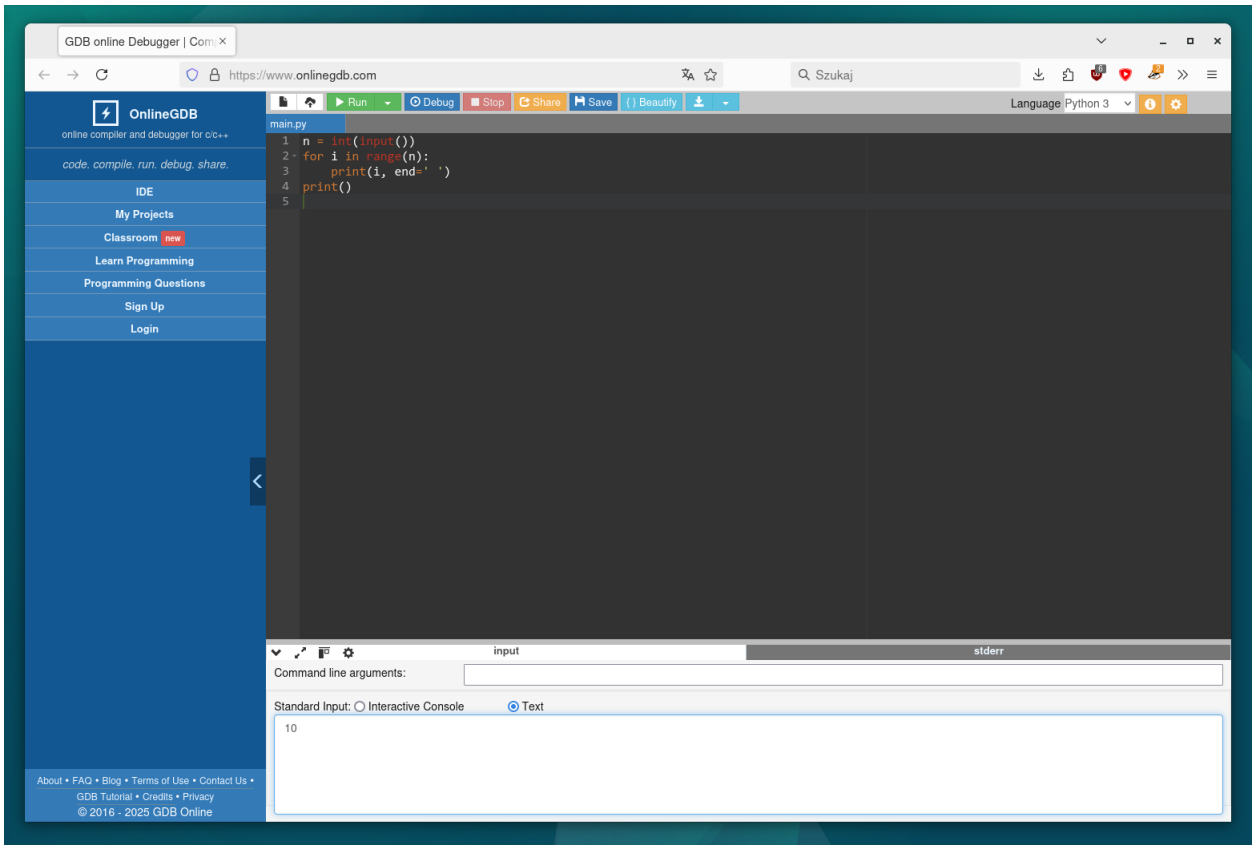


Gdyby nasz program zawierał błąd składniowy, zamiast wyniku działania moglibyśmy uzyskać informację o błędzie kompilacji programu.

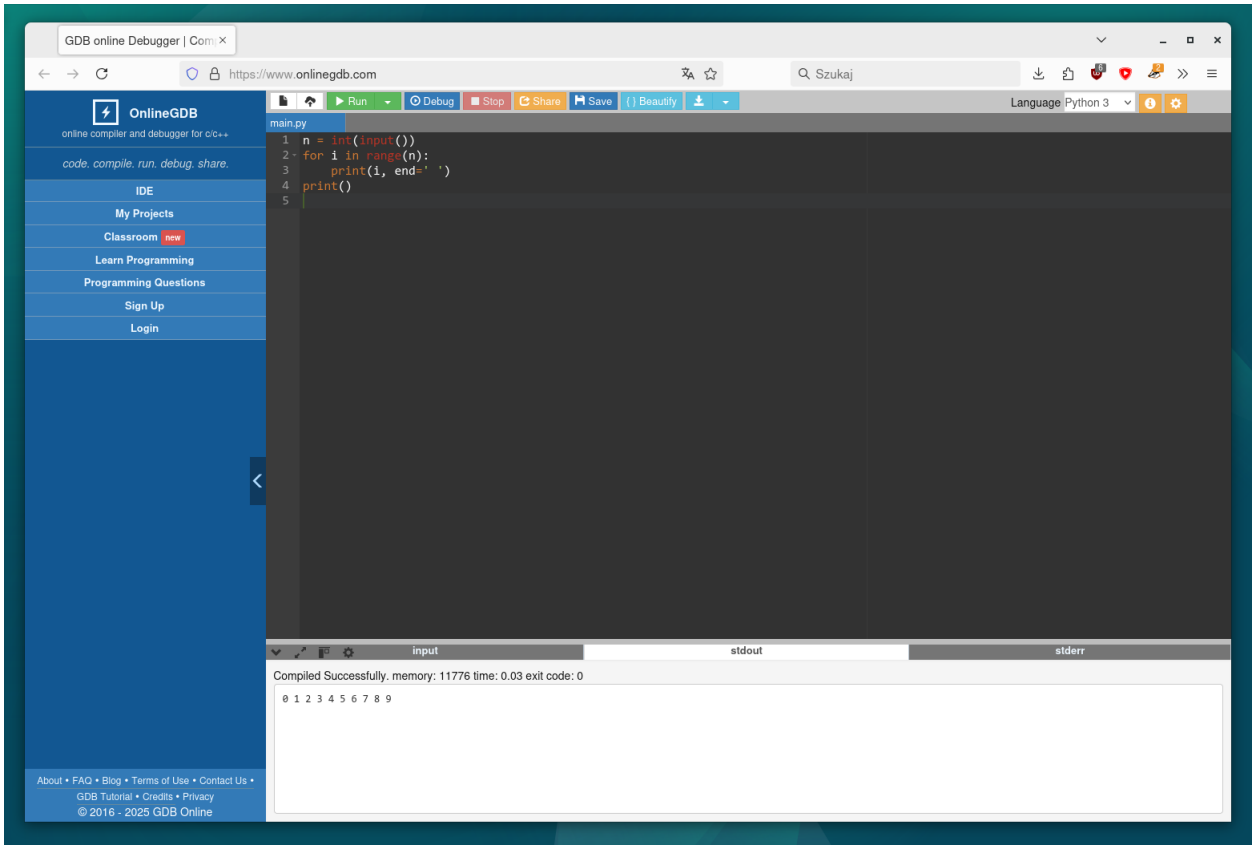
W poniższym przykładzie zapomnieliśmy średnika w linii 7, przez co utworzenie zmiennej `liczba` nie powiodło się. Po kliknięciu przycisku *Run* otrzymamy informację o błędzie. Możliwe jest poprawienie błędu i ponowne uruchomienie programu. Możliwe jest też przełączenie zakładki z `stderr` na `input` (w dolnej części okna) i wprowadzenie innych danych do poprawionego programu.



W analogiczny sposób możemy obsługiwać programy napisane w Pythonie. Należy pamiętać o wybraniu odpowiedniego języka z listy na górze po prawej (opcja *Language* powinna być ustawiona na **Python 3**). Dane wejściowe do programu również możemy wprowadzić w części na dole w zakładce `input`.

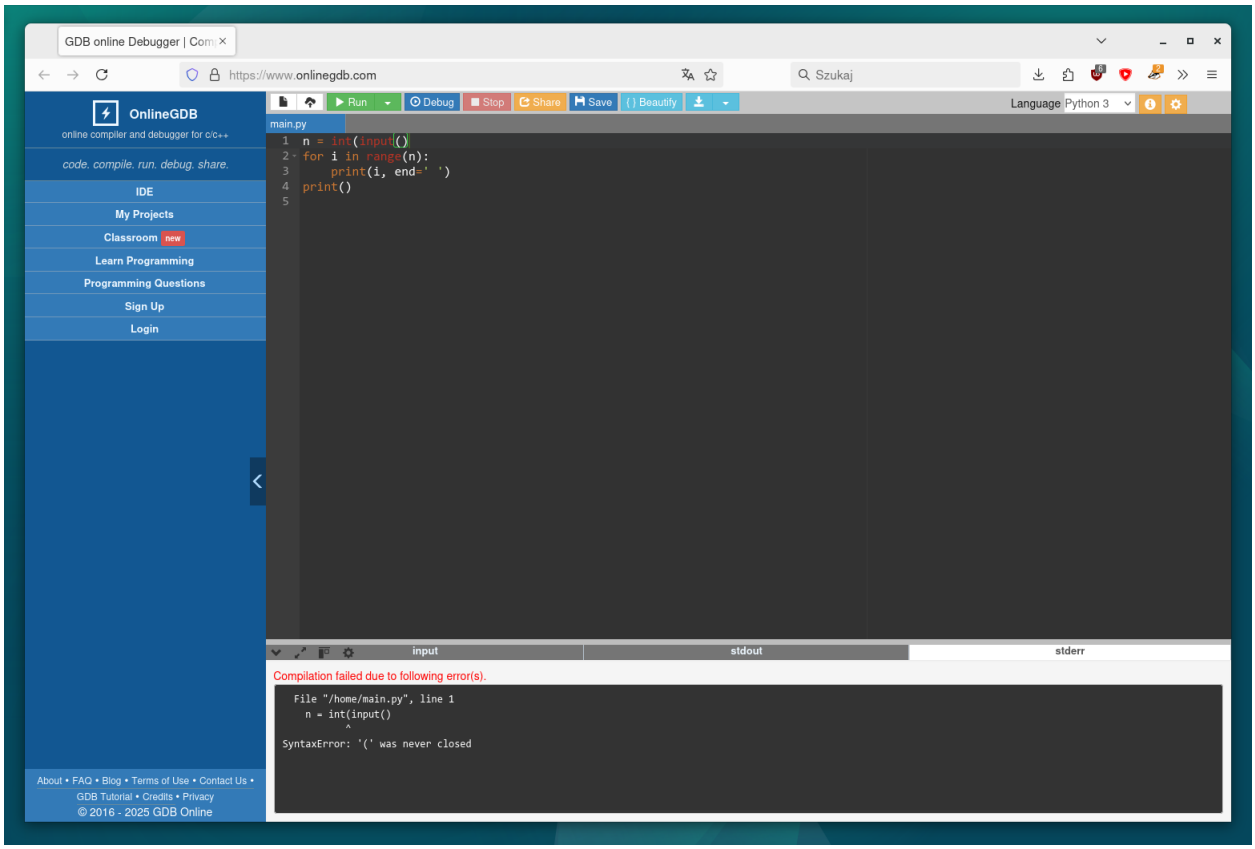


Uruchomienie programu również wykonujemy za pomocą przycisku *Run* w górnej części okna.



Ewentualne błędy składniowe lub błędy podczas uruchomienia programu zgłaszane są w dolnej części okna na zakładce `stderr`.

W poniższym przykładzie zapomnieliśmy o zamknięciu jednego nawiasu.



Uwaga: Możliwe jest pobranie swojego kodu na dysk (przycisk po prawej stronie od *Beautify*). W ten sposób możemy zrobić kopię zapasową naszej pracy. Należy bowiem pamiętać, że zamknięcie karty przeglądarki, w której znajduje się środowisko uruchomieniowe lub przeładowanie strony może spowodować utracenie pracy wewnątrz środowiska. Kopię zapasową można natomiast łatwo przywrócić ładując plik przyciskiem po lewej stronie od *Run* lub otwierając plik w notatniku i kopiując jego zawartość do okienka wewnątrz środowiska.

Dolnośląski Konkurs INFORMATYCZNY *zDolny Ślązak* dla uczniów szkół podstawowych w roku szkolnym 2025/2026		**ETAP POWIATOWY** **28 listopada 2025 r.** **godz. 12.00** czas trwania 90 minut
Kuratorium Oświaty we Wrocławiu / Dolnośląski Ośrodek Doskonalenia Nauczycieli we Wrocławiu		

INSTRUKCJA DO ARKUSZA PRZYKŁADOWEGO

1. Poniżej zapisuj swoje odpowiedzi. Oceniana jest tylko karta odpowiedzi na tej stronie.
2. W pytaniach zamkniętych z odpowiedziami (a), (b), (c), (d) do wyboru zakreśl znakiem **X** właściwą odpowiedź. W razie pomyłki otocz błędnie zaznaczoną odpowiedź kółkiem i jeszcze raz zaznacz poprawną odpowiedź. Poprawna jest zawsze dokładnie jedna z odpowiedzi.
3. Odpowiedzi w pozostałych pytaniach wpisz w pola w wyznaczonym miejscu.
4. Pamiętaj, że pracujesz samodzielnie. Możesz korzystać z komputera i przepisać, wzbogacać oraz uruchamiać podane w zadaniach programy. Możesz też korzystać z internetowego środowiska programistycznego lub kalkulatora systemowego. Nie możesz korzystać z żadnych innych pomocy – tablic, map, słowników, leksykonów, telefonów komórkowych, kalkulatorów kieszonkowych itp. **Nie możesz używać wyszukiwarki internetowej, szukać informacji na stronach internetowych, czy wykorzystywać komputera do komunikacji z innymi ludźmi lub narzędziami do generatywnej sztucznej inteligencji.** Potrzebne informacje zawarte są w treści zadań.
5. W arkuszu przykładowym znajduje się jedynie jedno zadanie. Maksymalna liczba punktów do zdobycia wynosi 10 – po 1 punkcie za każde pytanie. **Właściwy arkusz na etapie powiatowym będzie zawierał trzy zadania.** Korzystaj z przykładów ułatwiających zrozumienie zadań i spróbuj rozwiązać choć kilka pytań w każdym zadaniu. Nie musisz rozwiązywać po kolei – niektóre pytania są trudne i warto je ominąć, by najpierw zająć się łatwiejszymi pytaniami. Niektóre pytania można rozwiązać bez komputera, jedynie dzięki zrozumieniu algorytmów podanych w treści zadań, ale w niektórych pytaniach skorzystanie z komputera może być jedyną rozsądną metodą rozwiązania.
6. W niektórych przypadkach program dla danych z pytania będzie się wykonywał bardzo długo (dłużej niż czas trwania konkursu) lub będzie zużywał bardzo dużo pamięci (więcej niż ma dostępny komputer).

KARTA ODPOWIEDZI DO ARKUSZA PRZYKŁADOWEGO

1.1. (a) ☐ (b) ☐ (c) ☐ (d) ☐

1.2.

1.3.

1.4.

1.5.

1.6.

1.7.

1.8.

1.9.

1.10.

Przykładowe zadanie

Rozważmy następujący program (możesz założyć, że obie wersje są równoważne):

C++

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
  int n;
  cin >> n;
  vector<int> liczby;
  for (int i = 2; i <= n; i++) {
    bool czy_podzielna = false;
    for (int j : liczby)
      if (i % j == 0) {
        czy_podzielna = true;
        break;
      }
    if (!czy_podzielna)
      liczby.push_back(i);
  }
  for (int liczba : liczby)
    cout << liczba << "␣";
  return 0;
}
```

Python

```python
n = int(input())
liczby = []
for i in range(2, n + 1):
  czy_podzielna = False
  for j in liczby:
    if i % j == 0:
      czy_podzielna = True
      break
  if not czy_podzielna:
    liczby.append(i)
for liczba in liczby:
  print(liczba, end='␣')
```

Program przyjmuje na wejściu dodatnią liczbę całkowitą.

Przykład: Jeżeli do powyższego programu wprowadzić liczbę 10, to program wypisze następujące liczby (w tej kolejności, oddzielone spacjami): 2, 3, 5, 7.

? **1.1.** Co oblicza powyższy program?

(a) zbiór liczb nieparzystych mniejszych lub równych n,

(b) zbiór liczb pierwszych mniejszych lub równych n,

(c) zbiór liczb o takiej samej sumie cyfr co liczba n,

(d) zbiór wszystkich liczb naturalnych mniejszych lub równych n.

? **1.2.** Co wypisze powyższy program dla danej 20?

? **1.3.** Podaj przykład **dodatniej liczby całkowitej**, dla której powyższy program zwraca puste wyjście (nie wypisuje nic).

? **1.4.** Ile liczb **parzystych** wypisze powyższy program dla danej 500?

? **1.5.** Ile jest **dodatnich liczb całkowitych**, które podane na wejściu do powyższego programu spowodują, że wśród liczb wypisanych na wyjściu będzie liczba 1001?

? **1.6.** Ile liczb wypisze powyższy program dla danej 200?

? **1.7.** Ile wynosi suma liczb, które wypisze powyższy program dla danej 500?

? **1.8.** Ile wynosi **suma cyfr** liczb, które wypisze powyższy program dla danej 500?

? **1.9.** Podaj **najmniejszą dodatnią liczbę całkowitą**, która podana na wejściu do powyższego programu spowoduje wypisanie dokładnie 500 liczb.

? **1.10.** Ile liczb wypisze powyższy program dla danej 2 000 000 (dwa miliony)?

Przykładowe rozwiązanie

Rozważmy następujący program (możesz założyć, że obie wersje są równoważne):

C++

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
  int n;
  cin >> n;
  vector<int> liczby;
  for (int i = 2; i <= n; i++) {
    bool czy_podzielna = false;
    for (int j : liczby)
      if (i % j == 0) {
        czy_podzielna = true;
        break;
      }
    if (!czy_podzielna)
      liczby.push_back(i);
  }
  for (int liczba : liczby)
    cout << liczba << "␣";
  return 0;
}
```

Python

```python
n = int(input())
liczby = []
for i in range(2, n + 1):
  czy_podzielna = False
  for j in liczby:
    if i % j == 0:
      czy_podzielna = True
      break
  if not czy_podzielna:
    liczby.append(i)
for liczba in liczby:
  print(liczba, end='␣')
```

Program przyjmuje na wejściu dodatnią liczbę całkowitą.

Przykład: Jeżeli do powyższego programu wprowadzić liczbę 10, to program wypisze następujące liczby (w tej kolejności, oddzielone spacjami): 2, 3, 5, 7.

1.1. Co oblicza powyższy program?

(a) zbiór liczb nieparzystych mniejszych lub równych n,

(b) zbiór liczb pierwszych mniejszych lub równych n,

(c) zbiór liczb o takiej samej sumie cyfr co liczba n,

(d) zbiór wszystkich liczb naturalnych mniejszych lub równych n.

Rozwiązanie: Nawet nie rozumiejąc powyższego programu i jedynie analizując sam przykład (i odpowiedź do niego) można wywnioskować, że program ten generuje wszystkie liczby pierwsze mniejsze lub równe n.

Istotnie, najpierw wczytywana jest zmienna `n`, potem następuje pętla zewnętrzna, która testuje kolejne liczby naturalne od 2 do `n` włącznie. Następnie sprawdzana jest podzielność (operacja *modulo*) przez każdą z wcześniej wygenerowanych liczb. Jeżeli `n` nie dzieli się przez żadną z nich, to jest to liczba pierwsza i zostanie dodana na końcu ciągu `liczby`. Następnie program wypisuje zawartość ciągu `liczby` oddzielając elementy pojedynczymi odstępami.

1.2. Co wypisze powyższy program dla danej 20?

Rozwiązanie: Rozumiejąc już z poprzedniego pytania co robi ten program, nietrudno się domyślić, że program ten wypisuje liczby pierwsze nie większe niż 20. Są to 2, 3, 5, 7, 11, 13, 17, 19. Na karcie odpowiedzi należałoby je wypisać w tej kolejności, oddzielając pojedynczymi odstępami (bo dokładnie tak wypisze program).

Możliwe jest oczywiście również przepisanie powyższego programu do środowiska uruchomieniowego i wprowadzenie danej 20. Wówczas wystarczy jedynie przepisać na kartę odpowiedzi wyjście zwrócone przez środowisko.

1.3. Podaj przykład **dodatniej liczby całkowitej**, dla której powyższy program zwraca puste wyjście (nie wypisuje nic).

Rozwiązanie: Jedyną poprawną odpowiedzią jest liczba 1. Wprowadzenie większej liczby spowoduje wypisanie co najmniej dwójki, zaś wprowadzenie zera lub liczby ujemnej jest niedozwolone w treści pytania.

1.4. Ile liczb **parzystych** wypisze powyższy program dla danej 500?

Rozwiązanie: Jest tylko jedna liczba pierwsza, która jest parzysta (liczba 2). Oczywiście zostanie ona wypisana dla każdego wejścia większego lub równego 2. Odpowiedzią na pytanie jest więc 1.

1.5. Ile jest **dodatnich liczb całkowitych**, które podane na wejściu do powyższego programu spowodują, że wśród liczb wypisanych na wyjściu będzie liczba 1001?

Rozwiązanie: Liczba $1001 = 7 \cdot 11 \cdot 13$ nie jest liczbą pierwszą. Można się o tym przekonać na różne sposoby:

- przepisując powyższy program do środowiska uruchomieniowego i uruchomić go dla dowolnej danej większej lub równej 1002,
- implementując samodzielnie inny program, który sprawdza pierwszość liczby 1001,
- uruchamiając kalkulator systemowy i testując podzielność przez kilka pierwszych liczb pierwszych (wystarczy znaleźć jeden dzielnik właściwy, żeby odrzucić liczbę i uznać ją jako złożoną).

1.6. Ile liczb wypisze powyższy program dla danej 200?

Rozwiązanie: Jednym z pomysłów jest po prostu uruchomić program i zobaczyć odpowiedź i policzyć „ręcznie", ile liczb zostało wypisanych.

Tych liczb jednak jest sporo, dlatego lepszym pomysłem może być stworzyć zmienną `ile_liczb`, którą na początku programu ustawimy na 0, a instrukcję wypisywania liczby wymienić na zwiększenie tej zmiennej o 1.

Ewentualnie, zamiast tego, możemy wypisać długość ciągu `liczby`.

C++

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
  int n;
  cin >> n;
  vector<int> liczby;
  for (int i = 2; i <= n; i++) {
    bool czy_podzielna = false;
    for (int j : liczby)
      if (i % j == 0) {
        czy_podzielna = true;
        break;
      }
    if (!czy_podzielna)
      liczby.push_back(i);
  }
  cout << liczby.size();
  return 0;
}
```

Python

```python
n = int(input())
liczby = []
for i in range(2, n + 1):
  czy_podzielna = False
  for j in liczby:
    if i % j == 0:
      czy_podzielna = True
      break
  if not czy_podzielna:
    liczby.append(i)
print(len(liczby))
```

W każdy sposób powinniśmy uzyskać tę samą odpowiedź: 46.

? **1.7.** Ile wynosi suma liczb, które wypisze powyższy program dla danej 500?

Rozwiązanie: Jednym z pomysłów jest po prostu uruchomić program i zobaczyć odpowiedź i naiwnie obliczyć odpowiedź dodając do siebie wszystkie liczby z odpowiedzi za pomocą kalkulatora systemowego. Będzie to jednak zajmować cenny czas konkursu i zachodzi wysokie ryzyko pomyłki przy przepisywaniu liczb.

Innym pomysłem jest zmodyfikować program, aby na końcu zamiast wypisać te liczby, obliczyć ich sumę.

C++

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
  int n;
  cin >> n;
  vector<int> liczby;
  for (int i = 2; i <= n; i++) {
    bool czy_podzielna = false;
    for (int j : liczby)
      if (i % j == 0) {
        czy_podzielna = true;
        break;
      }
    if (!czy_podzielna)
      liczby.push_back(i);
  }
  int suma = 0;
  for (int liczba : liczby)
    suma += liczba;
  cout << suma;
  return 0;
}
```

Python

```python
n = int(input())
liczby = []
for i in range(2, n + 1):
  czy_podzielna = False
  for j in liczby:
    if i % j == 0:
      czy_podzielna = True
      break
  if not czy_podzielna:
    liczby.append(i)
suma = 0
for liczba in liczby:
  suma += liczba
print(suma)
```

Ostatecznie powinniśmy uzyskać sumę równą 21 536.

? **1.8.** Ile wynosi **suma cyfr** liczb, które wypisze powyższy program dla danej 500?

Rozwiązanie: Wzbogacamy powyższy program o funkcję obliczającą sumę cyfr, a następnie używamy tej funkcji, żeby obliczyć to, czego potrzebujemy.

C++

```
#include <iostream>
#include <vector>
using namespace std;

int suma_cyfr(int n) {
  int wynik = 0;
  while (n > 0) {
    wynik += n % 10;
    n /= 10;
  }
  return wynik;
}

int main() {
  int n;
  cin >> n;
  vector<int> liczby;
  for (int i = 2; i <= n; i++) {
    bool czy_podzielna = false;
    for (int j : liczby)
      if (i % j == 0) {
        czy_podzielna = true;
        break;
      }
    if (!czy_podzielna)
      liczby.push_back(i);
  }
  int suma = 0;
  for (int liczba : liczby)
    suma += suma_cyfr(liczba);
  cout << suma;
  return 0;
}
```

Python

```
def suma_cyfr(n):
  wynik = 0
  while n > 0:
    wynik += n % 10
    n //= 10
  return wynik

n = int(input())
liczby = []
for i in range(2, n + 1):
  czy_podzielna = False
  for j in liczby:
    if i % j == 0:
      czy_podzielna = True
      break
  if not czy_podzielna:
    liczby.append(i)
suma = 0
for liczba in liczby:
  suma += suma_cyfr(liczba)
print(suma)
```

Ostatecznie uzyskujemy wynik 1061.

? **1.9.** Podaj **najmniejszą dodatnią liczbę całkowitą**, która podana na wejściu do powyższego programu spowoduje wypisanie dokładnie 500 liczb.

Rozwiązanie: Możemy uruchomić program z pytania 1.6 dla jakiejś danej (na przykład 2000) i zobaczyć odpowiedź (ile liczb byłoby wypisanych, w tym przypadku 303 liczby). Jeżeli zostanie wypisane za mało, powiększamy wejście (np. próbujemy z daną 5000). Uzyskujemy wynik 669, a więc za dużo. Możemy więc (w podobny sposób jak w wyszukiwaniu binarnym) spróbować daną 3500. Otrzymamy 489 liczb pierwszych czyli niewiele mniej niż powinniśmy uzyskać. W ten sposób, po jeszcze kilku próbach możemy uzyskać odpowiedź do zadania: 3571.

Innym pomysłem jest dodać do programu z pytania 1.6 warunek, który przerywa wykonanie programu, gdy `liczby` będzie zawierać 500 elementów i uruchomić program dla dostatecznie dużej wartości zmiennej `n` (na przykład 5000).

Jeszcze innym pomysłem jest zamknąć program z pytania 1.6 w funkcję i naiwnie testować z użyciem brutalnej siły komputera coraz większe parametry.

C++

```
#include <iostream>
#include <vector>
using namespace std;

int ile_liczb(int n) {
  vector<int> liczby;
  for (int i = 2; i <= n; i++) {
    bool czy_podzielna = false;
    for (int j : liczby)
      if (i % j == 0) {
        czy_podzielna = true;
        break;
      }
    if (!czy_podzielna)
      liczby.push_back(i);
  }
  return liczby.size();
}

int main() {
  int n = 1;
  while (true) {
    if (ile_liczb(n) == 500) {
      cout << n;
      return 0;
    }
    n++;
  }
}
```

Python

```
def ile_liczb(n):
  liczby = []
  for i in range(2, n + 1):
    czy_podzielna = False
    for j in liczby:
      if i % j == 0:
        czy_podzielna = True
        break
    if not czy_podzielna:
      liczby.append(i)
  return len(liczby)

n = 1
while True:
  if ile_liczb(n) == 500:
    print(n)
    quit()
  n += 1
```

? **1.10.** Ile liczb wypisze powyższy program dla danej 2 000 000 (dwa miliony)?

Rozwiązanie: Naturalnym pomysłem byłoby uruchomić program z pytania 1.6 dla danej 2 000 000 zamiast dla danej 200. Okazuje się jednak, że podana w zadaniu implementacja jest bardzo niewydajna. Nie można oczekiwać, że program zakończy się w rozsądnym czasie dla takiej danej. Ale gdyby móc zaczekać odpowiednio długo, to program wypisałby jednak jakąś odpowiedź.

Należy obliczyć, ile jest liczb pierwszych mniejszych lub równych 2 000 000. Można to osiągnąć na przykład poprawiając implementację pokazaną w zadaniu.

C++

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
  int n;
  cin >> n;
  vector<int> liczby;
  for (int i = 2; i <= n; i++) {
    bool czy_podzielna = false;
    for (int j : liczby) {
      if (i % j == 0) {
        czy_podzielna = true;
        break;
      }
      // ponizej jest optymalizacja
      if (j * j > n) break;
    }
    if (!czy_podzielna)
      liczby.push_back(i);
  }
  cout << liczby.size();
  return 0;
}
```

Python

```
n = int(input())
liczby = []
for i in range(2, n + 1):
  czy_podzielna = False
  for j in liczby:
    if i % j == 0:
      czy_podzielna = True
      break
    # ponizej jest optymalizacja
    if j * j > n: break
  if not czy_podzielna:
    liczby.append(i)
print(len(liczby))
```

Możemy zastosować tę optymalizację, ponieważ każda liczba złożona n musi mieć dzielnik nie przekraczający $\sqrt{n}$. Istotnie, liczba złożona n musi się dać przedstawić jako iloczyn dwóch czynników p oraz q, takich, że p jest liczbą pierwszą oraz $p \leq q$. Gdyby p było większe od pierwiastka z n, to iloczyn $p \cdot q$ przekroczyłby n.

Tak napisany program powinien wykonać się w czasie kilku sekund.

Innym pomysłem (na nawet nieco szybszy program) jest zaimplementować znajdowanie liczb pierwszych metodą sita Eratostenesa.

C++

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
  int n;
  cin >> n;
  vector<bool> czy_zlozona(n+1, false);
  for (int i = 2; i * i <= n; i++) {
    if (czy_zlozona[i]) continue;
    for (int j = i * i; j <= n; j += i)
      czy_zlozona[j] = true;
  }
  int ile_pierwszych = 0;
  for (int i = 2; i <= n; i++)
    if (!czy_zlozona[i])
      ile_pierwszych++;
  cout << ile_pierwszych;
  return 0;
}
```

Python

```
n = int(input())
czy_zlozona = [False] * (n+1)
for i in range(2, int(n ** 0.5) + 1):
  if czy_zlozona[i]: continue
  for j in range(2 * i, n+1, i):
    czy_zlozona[j] = True
ile_pierwszych = 0
for i in range(2, n+1):
  if not czy_zlozona[i]:
    ile_pierwszych += 1
print(ile_pierwszych)
```

W obu przypadkach powinniśmy uzyskać tę samą odpowiedź: 148 933.