



Wydział Matematyki  
i Nauk Informacyjnych  
**Politechnika Warszawska**  
**KNSI Golem**



# KWANTOWA SZTUCZNA INTELIGENCJA

**dr hab. inż. Jerzy Balicki, prof. ucz.**

**Jerzy.Balicki@PW.EDU.PL**

Zakład Strukturalnych Metod Przetwarzania Wiedzy

**20 października 2022 r.**

# Motywacja

- Pierwsi wielcy twórcy Informatyki Norman Wiener, Alan Turing i John von Neumann opisywali wizje łączenia sztucznej inteligencji z obliczeniami kwantowymi.
- Obecnie te marzenia naukowe wydają się realne.
- Wprawdzie w najbliższych latach znacznie więcej oczekujemy po miniaturyzacji mikroprocesorów do nanoprocesorów oraz po skonstruowaniu sztucznego ludzkiego mózgu, tym niemniej fascynujący świat obliczeń kwantowych kusi możliwością rozwiązania problemów NP-trudnych czy osiągnięciem supremacji kwantowej w sztucznej inteligencji.

# Supremacja kwantowa

- Zainteresowanie algorytmami kwantowymi intensywnie rośnie.
- Zwłaszcza od 2019 roku, kiedy naukowcy z Google opublikowali pracę naukową, w której wykazali, że 53-kubitowy komputer Sycamore osiągnął supremację kwantową, czyli wyższą efektywność rozwiązywania problemów niż najmocniejszy superkomputer IBM Summit.
- Wg naukowców z Google, Sycamore w ciągu 200 sekund wykonał obliczenia, które komputerowi IBM Summit zajęłyby 10 tys. lat.
- Wprawdzie analiza wydajności komputerów przez naukowców z IBM nie potwierdziła aż takiej dominacji Sycamore, to nawet jeśli Summit potrzebowałby dwa i pół dnia na przeprowadzenie równoważnych obliczeń, a najszybszy w 2022 roku superkomputer Frontier ok. 4,5 godzin, to i tak można mówić o kolejnym przykładzie supremacji kwantowej.

# WĄTPLIWOŚCI

- Czy naprawdę pomyliłyśmy się, inwestując biliony dolarów w rozwój tradycyjnych komputerów?
- Czy nie będziemy żałować, jeśli wybierzemy informatykę kwantową, że zaprzestaliśmy rozwijać klasyczną informatykę?
- Co z milionami informatyków na całym świecie, którzy nie znają informatyki kwantowej?
- Studenci są w najlepszej sytuacji, ponieważ już teraz mogą poznać metody projektowania aplikacji kwantowych, a zwłaszcza aplikacji kwantowych implementujących algorytmy sztucznej inteligencji, w tym modele uczenia maszynowego.
- Wydaje się, że fascynacje i oczekiwania wielu z nas dotyczące eksploracji rozwoju kwantowych obliczeń w zakresie sztucznej inteligencji są olbrzymie.

# A JEŚLI KTOŚ NIE ZNA MECHANIKI KWANTOWEJ?

- **Kwantowa sztuczna inteligencja to nie mechanika kwantowa.** Proszę nie obawiać się zbyt złożonego aparatu matematycznego. Do tematu podchodzimy z obszaru Informatyki, aby nie utknąć w fizyce czy elektronice kwantowej – nie będziemy przecież budować komputerów kwantowych, ale raczej korzystać z dostępnego oprogramowania, komputerów kwantowych i symulatorów.
- **Oczywiście poznamy odpowiednie notacje, bramki kwantowe oraz podstawowe twierdzenia, ale skupiamy się przede wszystkim na algorytmach i modelach sztucznej inteligencji, które możemy zaimplementować na komputerach kwantowych, uruchomić w symulatorze kwantowym lub na tradycyjnym komputerze w dobrze znanym środowisku.**

# WPROWADZENIE DO KSI

- 1. Wprowadzenie do komputerów kwantowych; Bramki jedno, dwu i trzykubitowe; bramka Hadamarda, Feynmana i Toffoliego; Superpozycja, splątanie i teleportacja kwantowa.**
- 2. Algorytmy kwantowe; reprezentacja rejestrów kwantowych; algorytm Deutscha-Jozsy, algorytm Shora, algorytm Grovera, sumator kwantowy, korekcja błędów;**
- 3. Supremacja kwantowa; Symulatory kwantowe;**
- 4. Kwantowo-inspirowane sztuczne sieci neuronowe;**
- 5. Kwantowe algorytmy genetyczne i ewolucyjne;**

# Zaawansowana KSI

- 6. Kwantowe algorytmy PSO, ACO i ABC;**
- 7. Kwantowe uczenie maszynowe;**
- 8. Kwantowe modele Deep Learningu;**
- 9. Zasady implementacji algorytmów kwantowych w wybranych środowiskach;**
- 10. Algorytmy kwantowe w cyberbezpieczeństwie.**

# Projektowanie aplikacji

Implementacje wybranych algorytmów kwantowych w zakresie sztucznej inteligencji mogą wykorzystywać:

- a) serwisy chmur obliczeniowych,
- b) komputery kwantowe o ograniczonej liczbie kubitów,
- c) symulatory kwantowe: Quantum Inspire (QuTech), IBM Quantum Lab,
- d) biblioteki bramek kwantowych i modeli sztucznej inteligencji: Qiskit (IBM), QASM Simulator (Qiskit), Cirq (Google), TensorFlow Quantum (Google),
- e) Języki programowania: Python, cQASM (QuTech), Q# (Microsoft)

# Porównanie środowisk Quantum Inspire i Cirq



- + przejrzysty interfejs
- + baza wiedzy z przykładami
- + możliwy dostęp do prawdziwego QPU
- + posiada API
- wymaga stałego połączenia sieciowego
- ograniczony dostęp do backendów

- + zaimplementowany jako biblioteka do Pythona
- + możliwość symulacji na własnym komputerze
- + duży wybór backendów (nie tylko symulatorów)
- dużo bardziej skomplikowany
- symulacja na własnym urządzeniu może wymagać złożonej specyfikacji urządzenia

# Porównanie efektywności środowisk Quantum Inspire i Cirq

- **Algorytm Grovera uruchomiono jako skrypt w języku cQASM w symulatorze Quantum Inspire oraz jako skrypt w języku Python korzystający z biblioteki Cirq w Google Colab;**
- **Kwantowy algorytm Grovera służy do przeszukiwania baz danych;**
- **Złożoność obliczeniowa to  $O(\sqrt{n})$ , złożoność układu kwantowego  $O(N \log N)$  (liczba bramek), prawdopodobieństwo wyznaczenia poprawnego wyniku  $p \geq 2/3$ .**

# Kwantowy algorytm Grovera

1. Inicjalizacja kubitów w stan  $|0\rangle$  oraz w stan superpozycji;
2. Poszukiwanie  $x_0$  na podstawie wartości funkcji  $f$ ;
3. Zastosowanie operatora dyfuzji Grovera (inwersja o średnią);
4. Powtórzenia kroku 2 i 3;
5. Pomiar końcowy.

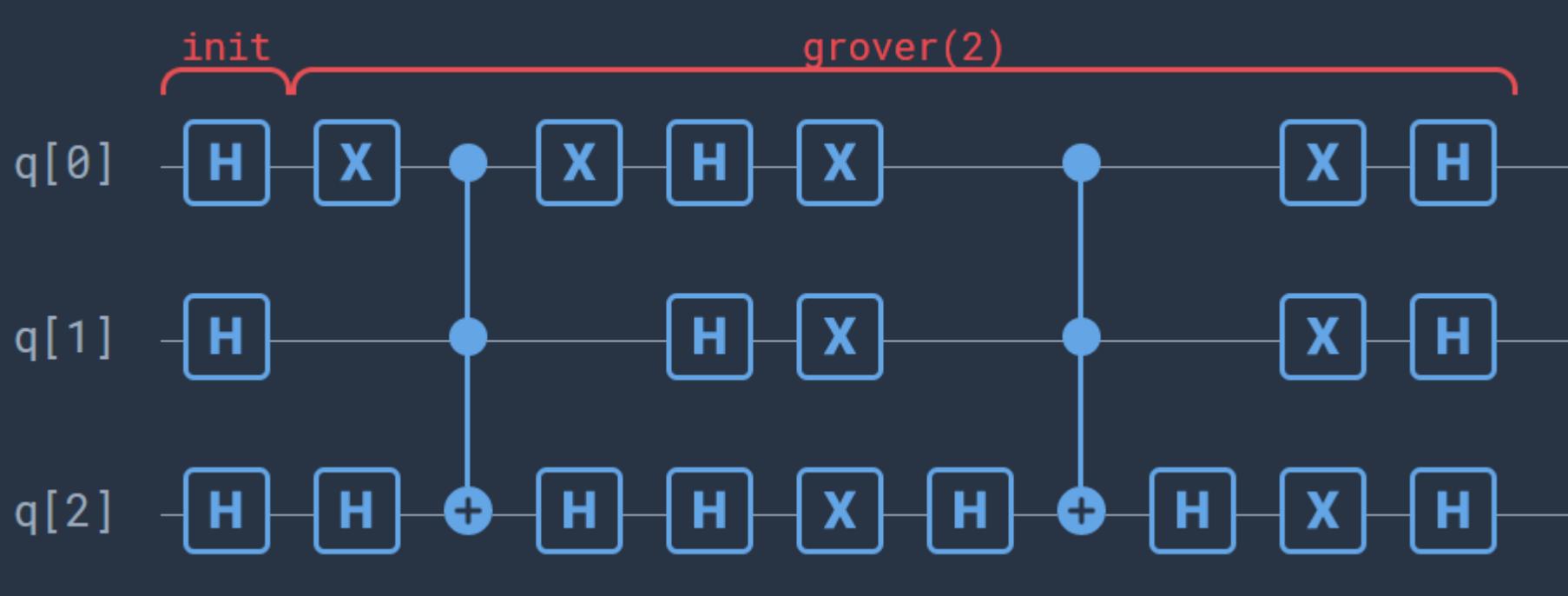
# Kwantowy algorytm Grovera

```
1 version 1.0
2 qubits 3
3 # Grover's algorithm for searching the decimal number 6 in a database
4 .init
5 H q[0:2]
6 .grover(2)
7 # oracle
8 {X q[0] | H q[2] }
9 Toffoli q[0], q[1], q[2]
10 {H q[2] | X q[0]}
11
```

# Kwantowy algorytm Grovera

```
11
12 # diffusion
13 {H q[0] | H q[1] | H q[2]}
14 {X q[1] | X q[0] | X q[2] }
15 H q[2]
16 Toffoli q[0], q[1], q[2]
17 H q[2]
18 {X q[1] | X q[0] | X q[2] }
19 {H q[0] | H q[1] | H q[2]}
20
21 # Measurement not required on emulator backend
```

# Kwantowy algorytm Grovera



# Kwantowy algorytm Grovera

## Run Experiment

Amount of qubits: 3

Backend: QX-34-L (34-qubit)

Experiment name

New Experiment

Number of shots

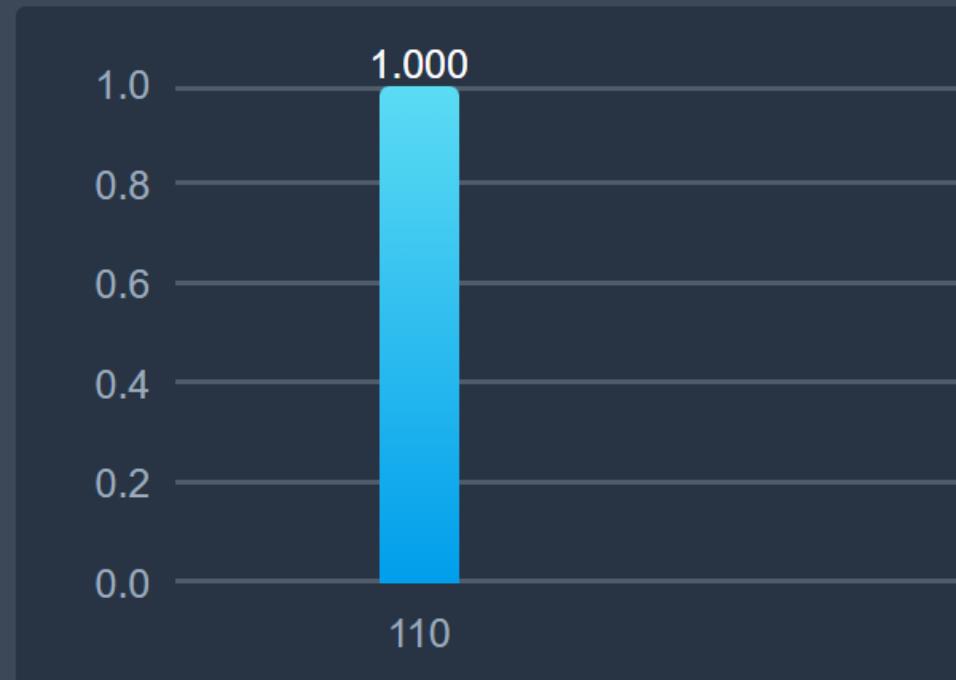
2

⚠ This run will be optimized. [Learn more ↗](#)

Cancel

Run

Device:	QX-34-L
Number of shots:	2
Execution time:	0.066s



Uruchamiamy samodzielnie: <https://www.quantum-inspire.com/kbase/grover-algorithm/>

# Implementacja algorytmu Grovera z wykorzystaniem biblioteki Cirq (Python, Google Colab)

try:

```
    import cirq
```

except ImportError:

```
    print("installing cirq...")
```

```
    !pip install cirq --quiet
```

```
    import cirq
```

```
    print("installed cirq.")
```

```
import random
```

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

# Konstrukcja rejestru kwantowego

```
"""Konstrukcja rejestru kwantowego"""
# Liczba kubitów n=2.
nqubits = 2

# Konstrukcja rejestru oraz kubitu pomocniczego (ancilla).
qubits = cirq.LineQubit.range(nqubits)
ancilla = cirq.NamedQubit("Ancilla")
```

# Implementacja funkcji pomocniczej make\_oracle

```
def make_oracle(qubits, ancilla, xprime):
    """Implementacja funkcji {f(x) = 1 if x == x', f(x) = 0 if x != x'}."""
    # Dla x' = (1, 1), funkcja jest bramką Toffoli.
    # Dla dowolnego x', bity |0> są negowane i jest bramka Toffoli.
    # Jeśli konieczne, neguj bity |0>.
    yield (cirq.X(q) for (q, bit) in zip(qubits, xprime) if not bit)
    # Wykonuj bramkę Toffoli.
    yield (cirq.TOFFOLI(qubits[0], qubits[1], ancilla))
    # Jeśli konieczne, neguj bity |0>
    yield (cirq.X(q) for (q, bit) in zip(qubits, xprime) if not bit)
```

# Implementacja funkcji jednokrotnego wywołania układu Grovera ( $n=2 \Rightarrow 1$ iteracja)

```
def grover_iteration(qubits, ancilla, oracle):
    """Wykonaj jedną iterację układu Grovera (n=2)."""
    circuit = cirq.Circuit()
    # Ustawienie superpozycji stanów kubitów wejściowych za pomocą bramek Hadamarda.
    circuit.append(cirq.H.on_each(*qubits))
    # Ustawienie kubitu wyjściowego w stan  $|-\rangle$ .
    circuit.append([cirq.X(ancilla), cirq.H(ancilla)])
    # Wywołaj wyrocznię (the oracle).
    circuit.append(oracle)
    # Konstrukcja układu (operator) Grovera.
    circuit.append(cirq.H.on_each(*qubits))
    circuit.append(cirq.X.on_each(*qubits))
    circuit.append(cirq.H.on(qubits[1]))
    circuit.append(cirq.CNOT(qubits[0], qubits[1]))
    circuit.append(cirq.H.on(qubits[1]))
    circuit.append(cirq.X.on_each(*qubits))
    circuit.append(cirq.H.on_each(*qubits))
    # Pomiar rejestru wejściowego.
    circuit.append(cirq.measure(*qubits, key="result"))
return circuit
```

# Konstrukcja układu Grovera

```
"""Losowy wybór 'oznaczonego' łańcucha kubitów x'."""
```

```
xprime = [random.randint(0, 1) for _ in range(nqubits)]
```

```
print(f"Marked bitstring: {xprime}")
```

```
Marked bitstring: [1, 0]
```

```
"""Konstrukcja układu kwantowego dla algorytmu Grovera."""
```

```
# Konstrukcja wyroczni - wywołanie funkcji make_oracle
```

```
oracle = make_oracle(qubits, ancilla, xprime)
```

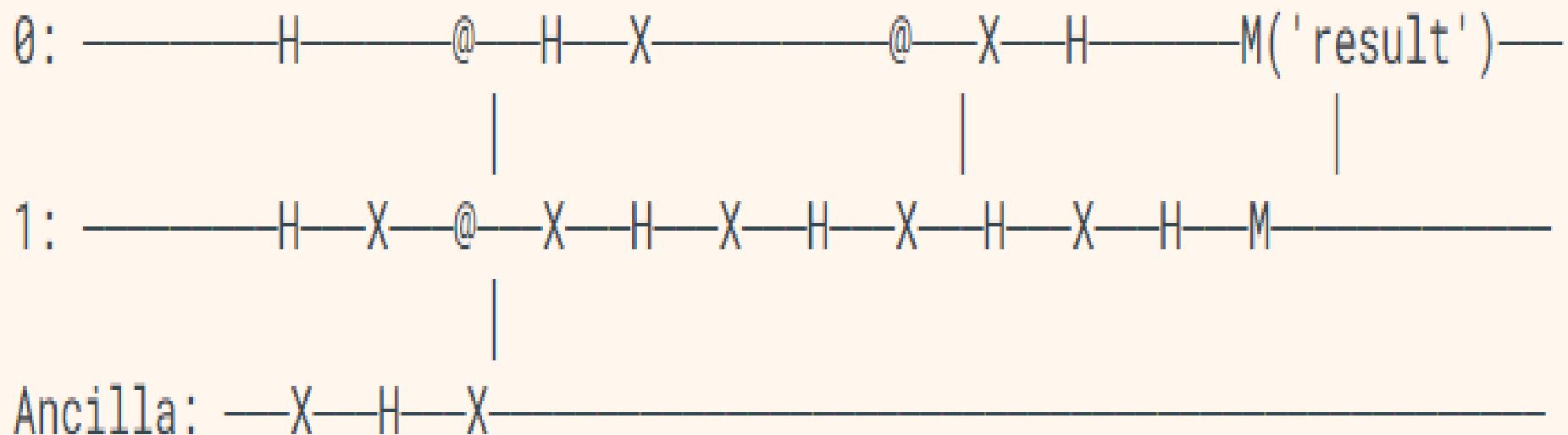
```
# Wstawienie wyroczni do układu Grovera.
```

```
circuit = grover_iteration(qubits, ancilla, oracle)
```

```
print("Circuit for Grover's algorithm:")
```

```
print(circuit)
```

# Układ Grovera



# Symulacja układu Grovera

```
"""Symulacja układu Grovera i odczyt wyników."""
# Funkcja pomocnicza.
def bitstring(bits):
    return "".join(str(int(b)) for b in bits)
# Wielokrotna symulacja układu, 10 razy.
simulator = cirq.Simulator()
result = simulator.run(circuit, repetitions=10)
# Wyświetlenie wyników.
frequencies = result.histogram(key="result", fold_func=bitstring)
print('Sampled results:\n{}'.format(frequencies))
# Check if we actually found the secret value.
most_common_bitstring = frequencies.most_common(1)[0][0]
print("\nMost common bitstring: {}".format(most_common_bitstring))
print("Found a match? {}".format(most_common_bitstring ==
bitstring(xprime)))
```

# Wyniki

```
Sampled results:
```

```
Counter({'10': 10})
```

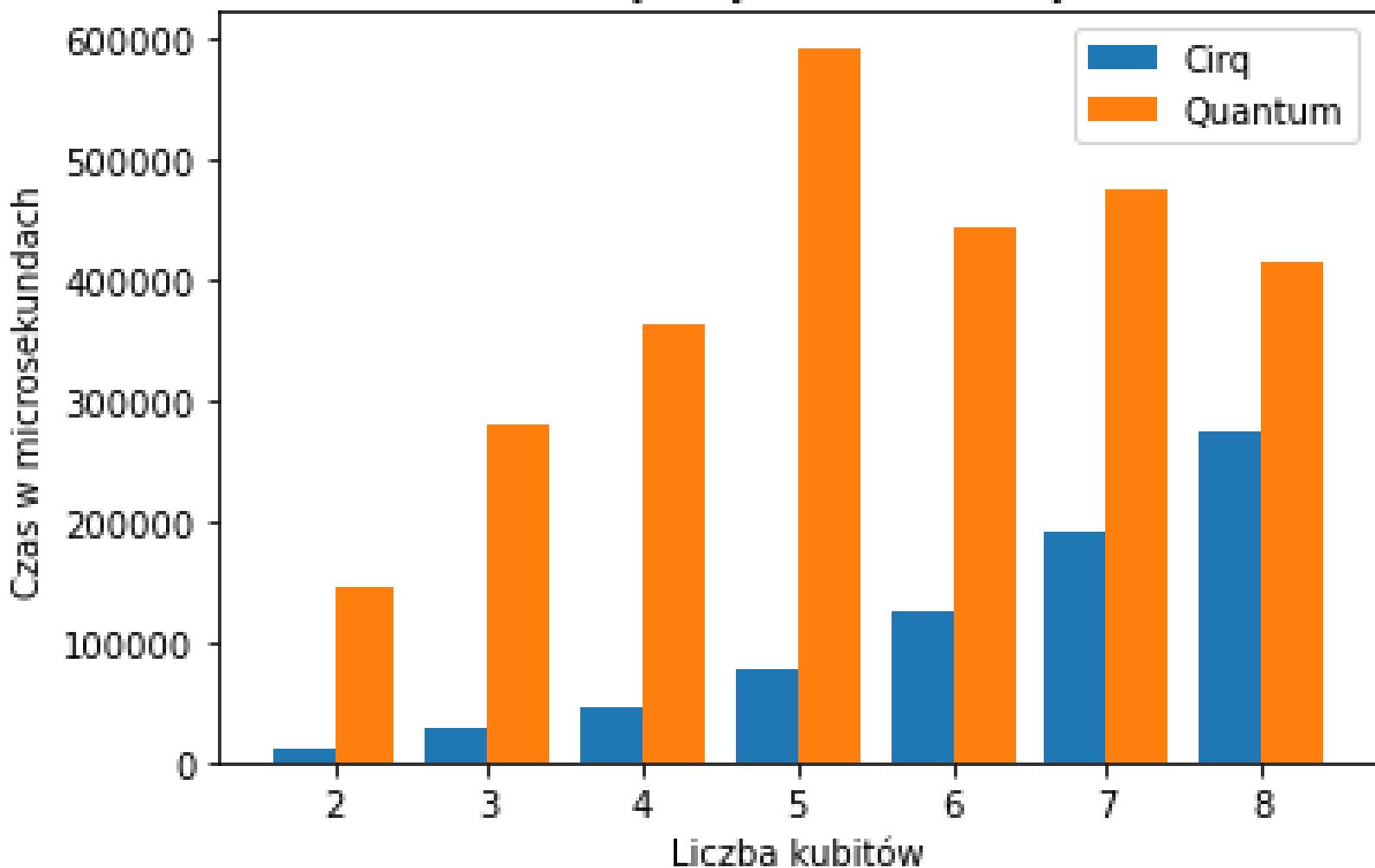
```
Most common bitstring: 10
```

```
Found a match? True
```

Uruchamiamy samodzielnie:

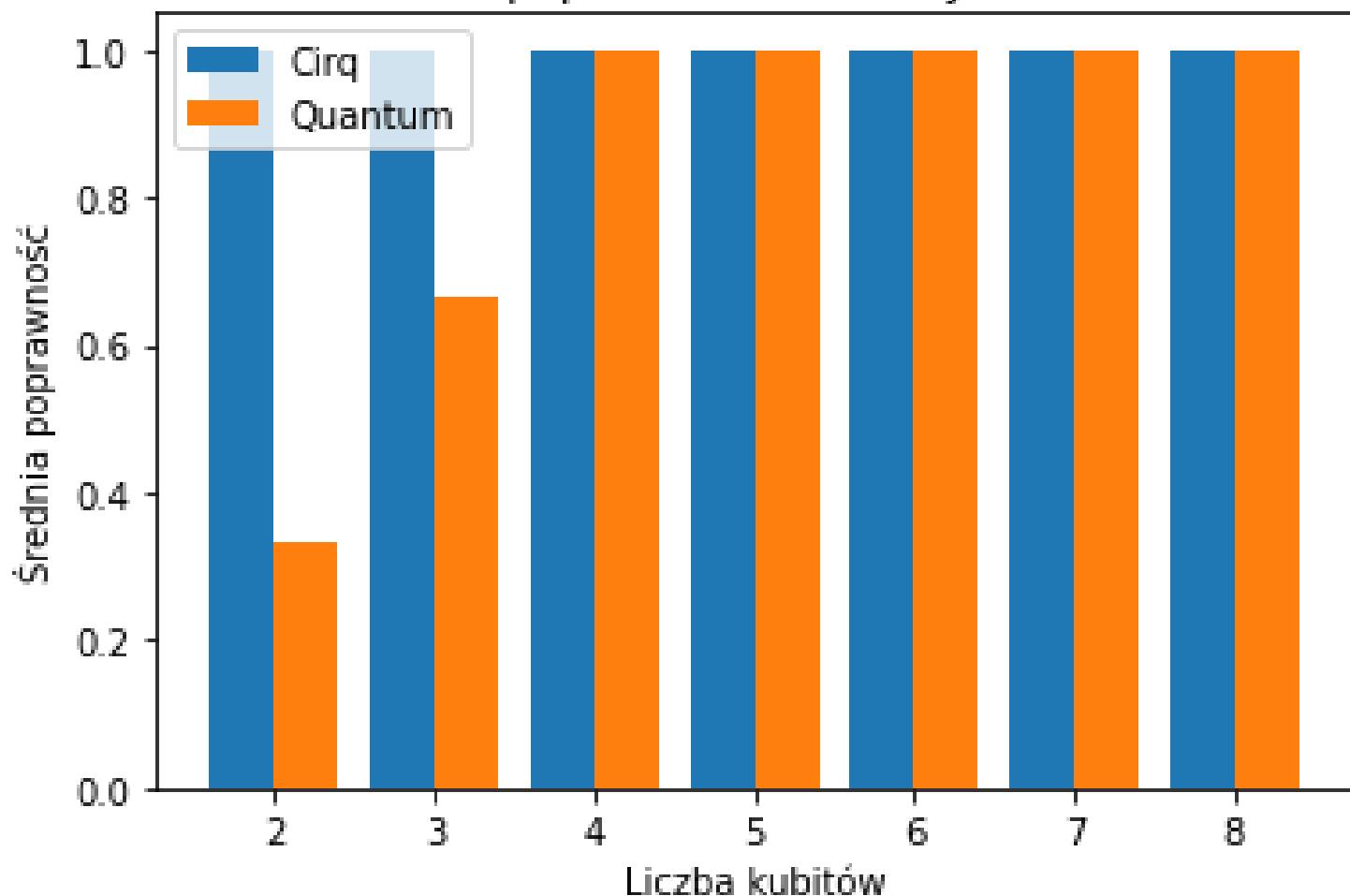
[https://quantumai.google/cirq/experiments/textbook\\_algorithms](https://quantumai.google/cirq/experiments/textbook_algorithms)

## Średni czas wykonywania od liczby kubitów



By Anna Buchman, Michał Machnio, Miłosz Mazur, Adam Skaskiewicz

## Średni poprawność od liczby kubitów



By Anna Buchman, Michał Machnio, Miłosz Mazur, Adam Skaskiewicz

# Simulator IBM Quantum Lab

The screenshot shows the IBM Quantum Lab interface with a Jupyter Notebook open. The left sidebar displays a file tree with 'Lab files /' containing 'qiskit-textbook', 'qiskit-tutorials', 'Untitled.ipynb', and 'Untitled1.ipynb'. The right pane shows the notebook interface with a toolbar above and a code cell below. The code cell contains Python code for importing Qiskit libraries and providers, with the 'QasmSimulator' import statement highlighted by a red oval.

```
[1]: import numpy as np

# Importing standard Qiskit libraries
from qiskit import QuantumCircuit, transpile, Aer, IBMQ
from qiskit.tools.jupyter import *
from qiskit.visualization import *
from ibm_quantum_widgets import *
from qiskit.providers.aer import QasmSimulator

# Loading your IBM Quantum account(s)
provider = IBMQ.load_account()

<frozen importlib._bootstrap>:219: RuntimeWarning: scipy._lib.messagestream.MessageStream size changed, may indicate binary incompatibility. Expected 56 from C header, got 64
from PyObject
```

By Alicja Dailczuk, Martin Mrugała, Filip Szymczak i Patryk Walczak

# Cirq vs. Qiskit dla algorytmu Shora faktoryzacji liczb

Rodzaj algorytmu	Przetwarzana liczba	Średni czas przetwarzania [ms]
cirq-quantum	15	1473.9976
cirq-quantum	21	29868.0233
cirq-quantum	30	865.5668
qiskit	9	716.3219
qiskit	15	789.3233
qiskit	18	721.6285
qiskit	21	988.1656
qiskit	25	1107.3533
qiskit	30	779.0965

# Kwantowe sieci neuronowe QNN w TensorFlow Quantum

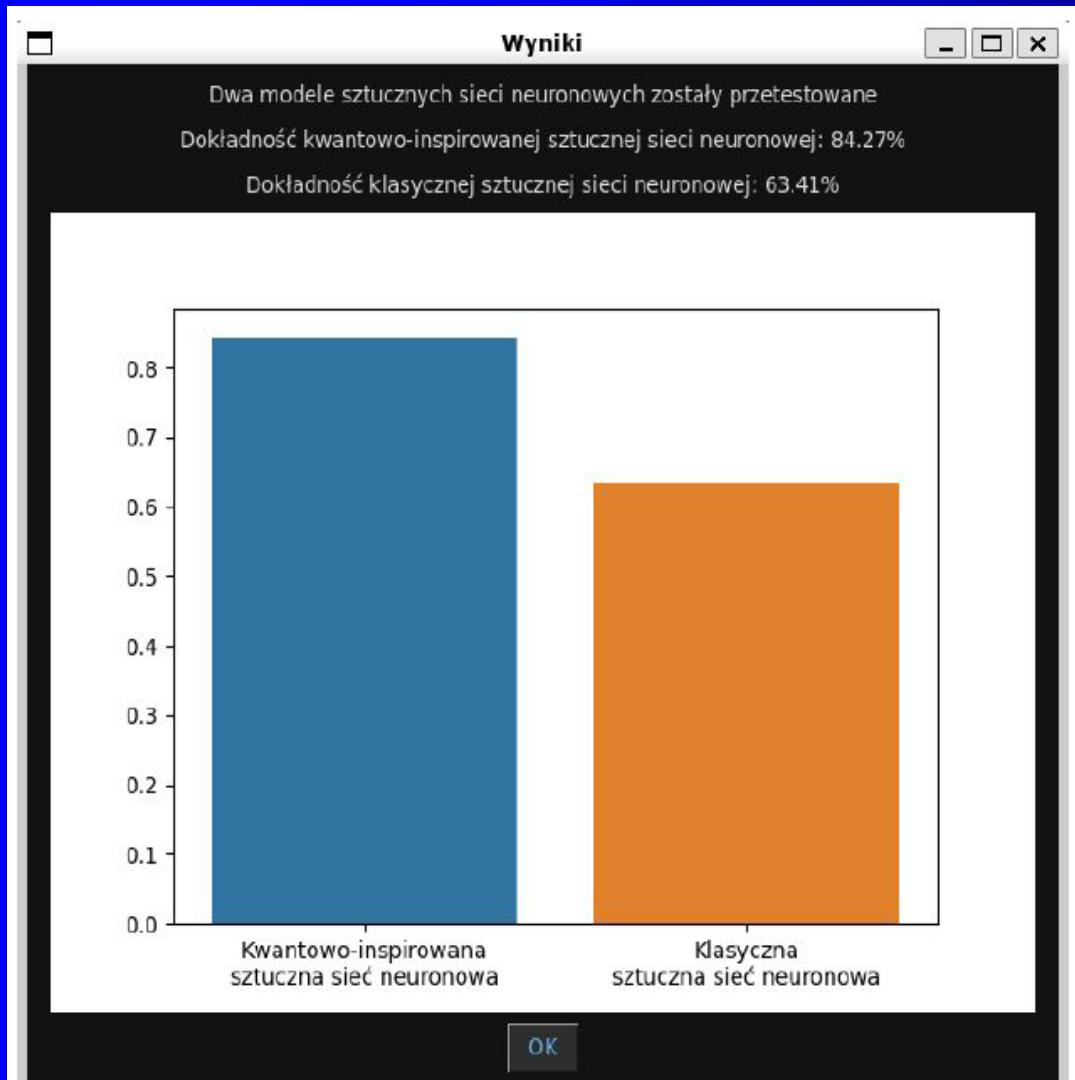
1. Dane wejściowe kodowane są na odpowiadające im stanów kubitowe,
2. Kubity są transformowane za pomocą bramek rotacyjnych,
3. Kubity wprowadzane są w stan splątania,
4. Warstwa neuronów składa się z bramek rotacyjnych i bramek generujących splątania kubitów. Na wyjściu sieci stan wyjścia jest mierzony,
5. Dekodowanie stanów kubitów na format wynikowy.

# QNN z TensorFlow Quantum

## Drobne problemy:

- Konieczność użycia Linuksa lub WSL, które jest dostępne wyłącznie w Windows 10 i 11;
- Nie wszystkie procesory wspierają wirtualizację, niezbędną do działania WSL;
- Konieczność ręcznej instalacji i konfiguracji X-serwera;
- Najnowsze wersje Pythona mogą być niekompatybilne.

# Dokładność QNN i ANN dla MNIST

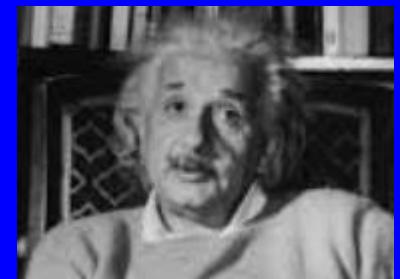


- Rozmiar batcha: 64
- Liczba epok: 3
- Dokładność QNN: 84.27%
- Dokładność CNN: 63.41%

# Kwantowość a sztuczna inteligencja

# Inteligencja a intelekt

**Intelekt** – całokształt wiedzy, ogólny doświadczenie i zdolności umysłowych człowieka (przysługuje tylko człowiekowi)



**Inteligencja** – termin definiowany niejednolicie i często utożsamiany z pojęciem intelekt.

*Zespół zdolności umożliwiający jednostce na korzystanie z nabytej wiedzy oraz skuteczne zachowanie się wobec nowych zadań i warunków.*



# ***Współczesna definicja inteligencji***

**Inteligencja** – zdolność przystosowania się do środowiska i okoliczności za pomocą:

- dostrzegania abstrakcyjnych relacji,
- korzystania ze zdobytych doświadczeń,
- skutecznej kontroli nad własnymi procesami poznaowczymi.



# Sztuczna Inteligencja - AI

- Pojęcie sztucznej inteligencji (ang. *Artificial Intelligence* - AI) pojawiło się w połowie lat pięćdziesiątych XX wieku:

*McCarthy J., Programs with common sense. In: Mechanization of Thought Processes. HMSO, London 1950, pp.75-91.*



**AI to kierunek zastosowań komputerów do rozwiązywania problemów, których formułowanie i rozwiązywanie uznawane było wcześniej za wyłączną domenę człowieka.**

# Trzy kryteria akceptacji aplikacji cechującej się sztuczną inteligencją

- Symulacja procesów naturalnych (z użyciem testu Turinga);
- Inteligentne czynności;
- Racjonalne sprawstwo.

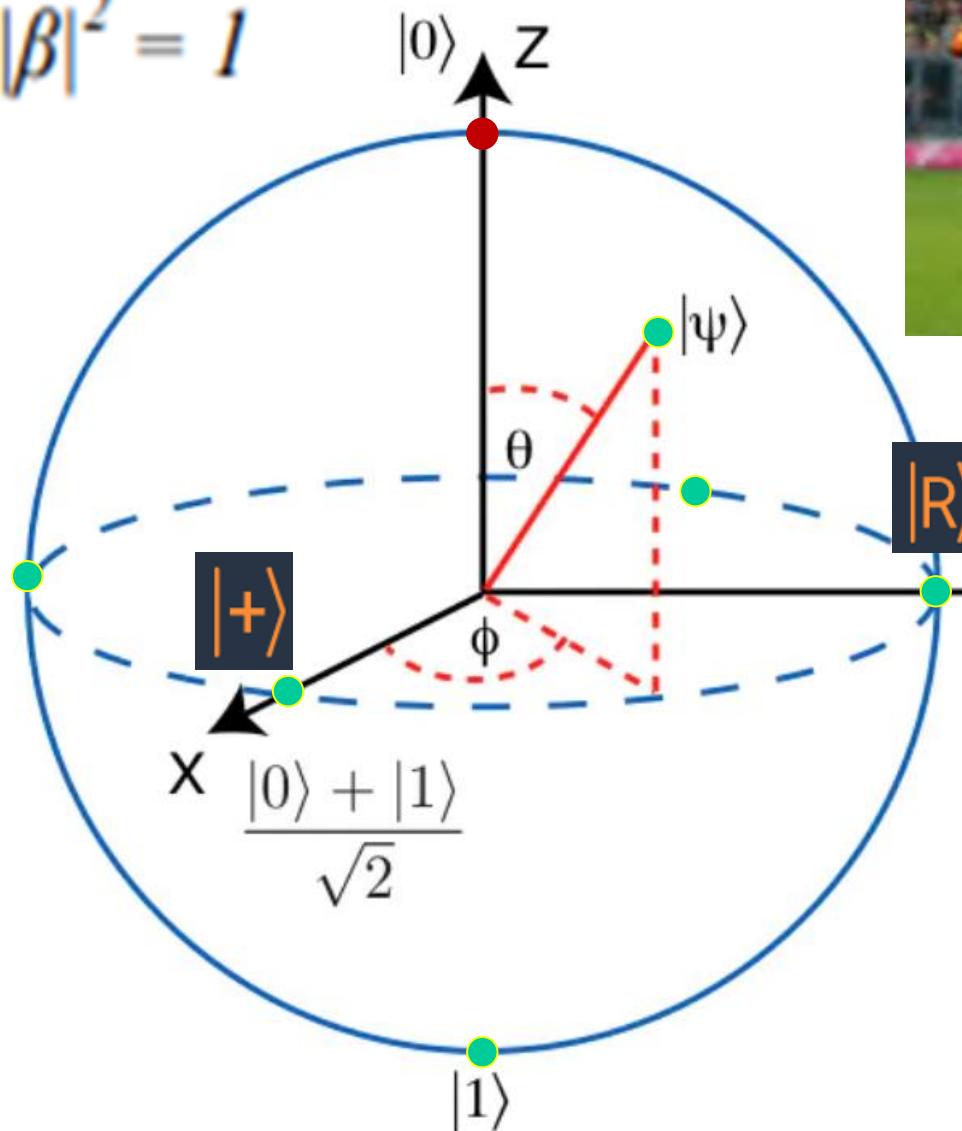


# Dziedzina AI

- rozwiązywanie złożonych problemów,
- strategie przeszukiwań,
- teoria gier,
- sztuczne sieci neuronowe,
- algorytmy ewolucyjne,
- automatyczne dowodzenie twierdzeń,
- przetwarzanie języka naturalnego,
- systemy ekspertowe,
- procesy percepacji,
- uczenie maszynowe,
- wyszukiwanie informacji (inteligentne bazy danych),
- programowanie automatyczne,
- logika rozmyta
- hurtownie danych ...

# Inicjowanie pojedynczego kubitu qubits 1

$$|\alpha|^2 + |\beta|^2 = 1$$



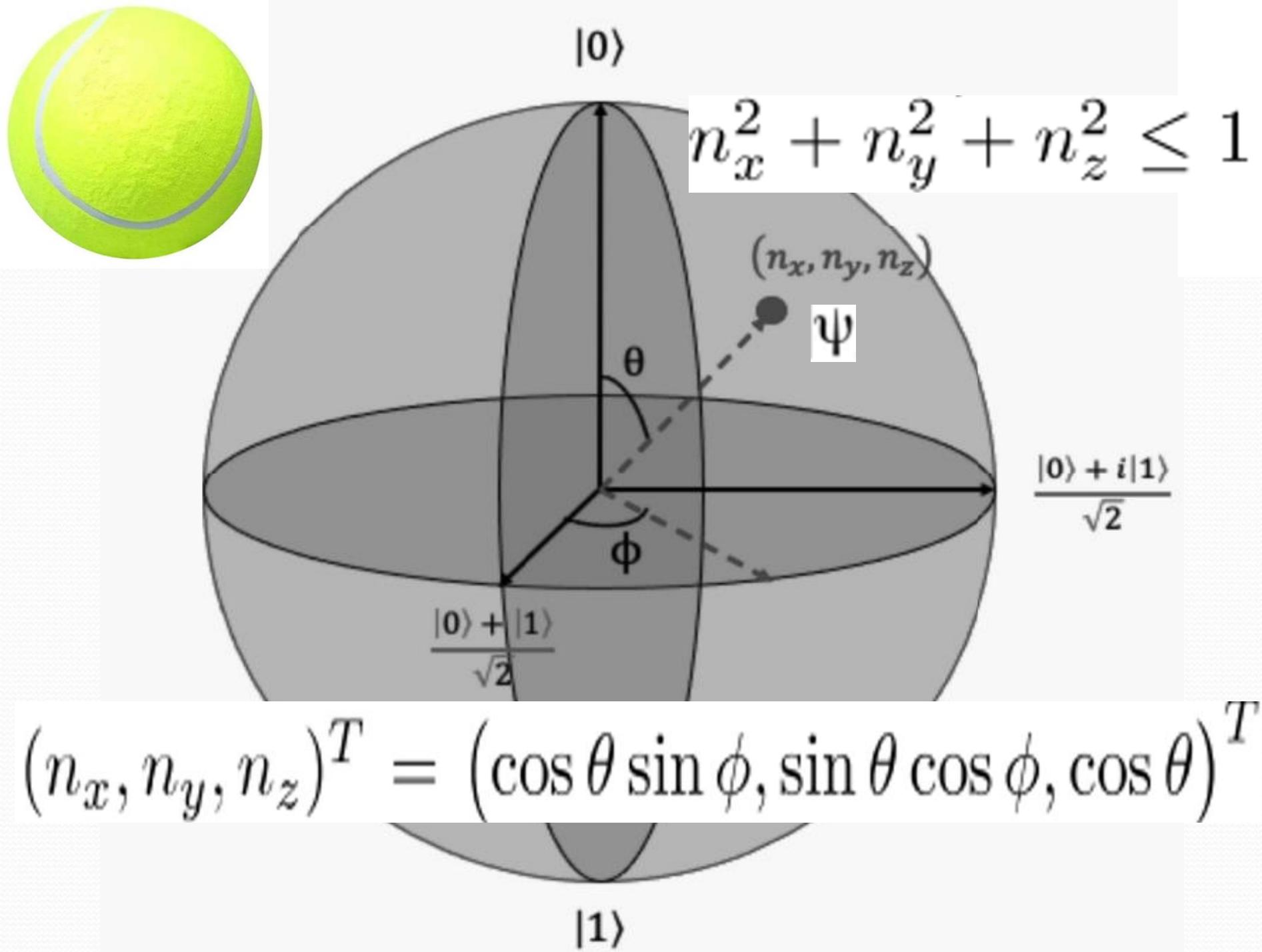
$$|\Psi\rangle = \alpha |0\rangle + \beta |1\rangle$$

$$\frac{|0\rangle + i|1\rangle}{\sqrt{2}}$$

$|\Psi\rangle$  - psi

# Czy Iga Świątek wygra Roland Garros w 2023?





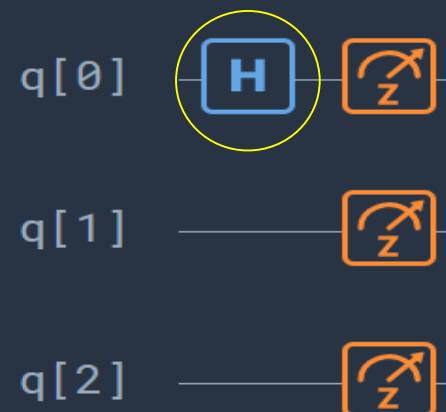
# Wprowadzenie wybranego kubitu w stan superpozycji (pozostałe równe zeru)



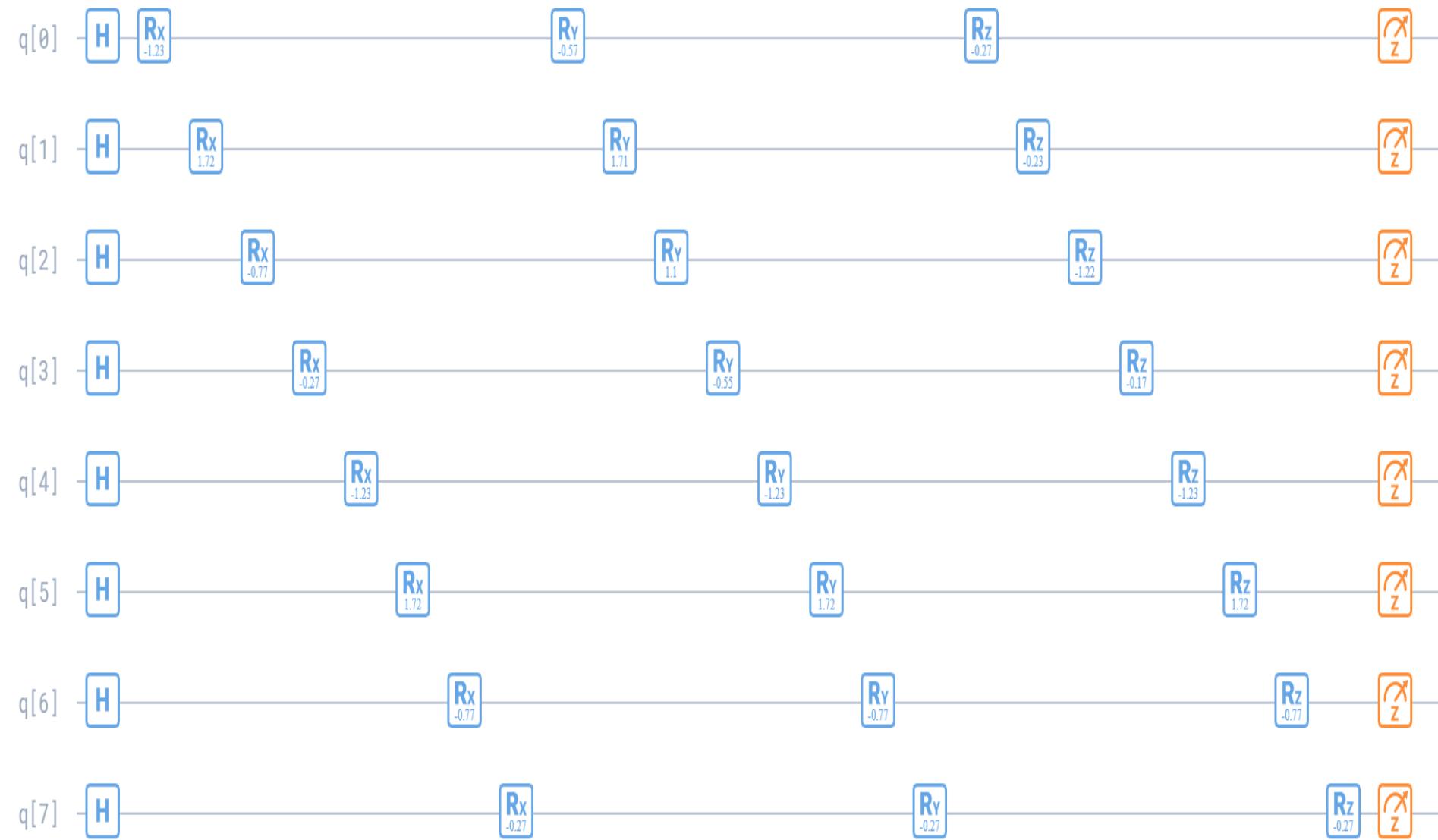
Superpozycja RL9



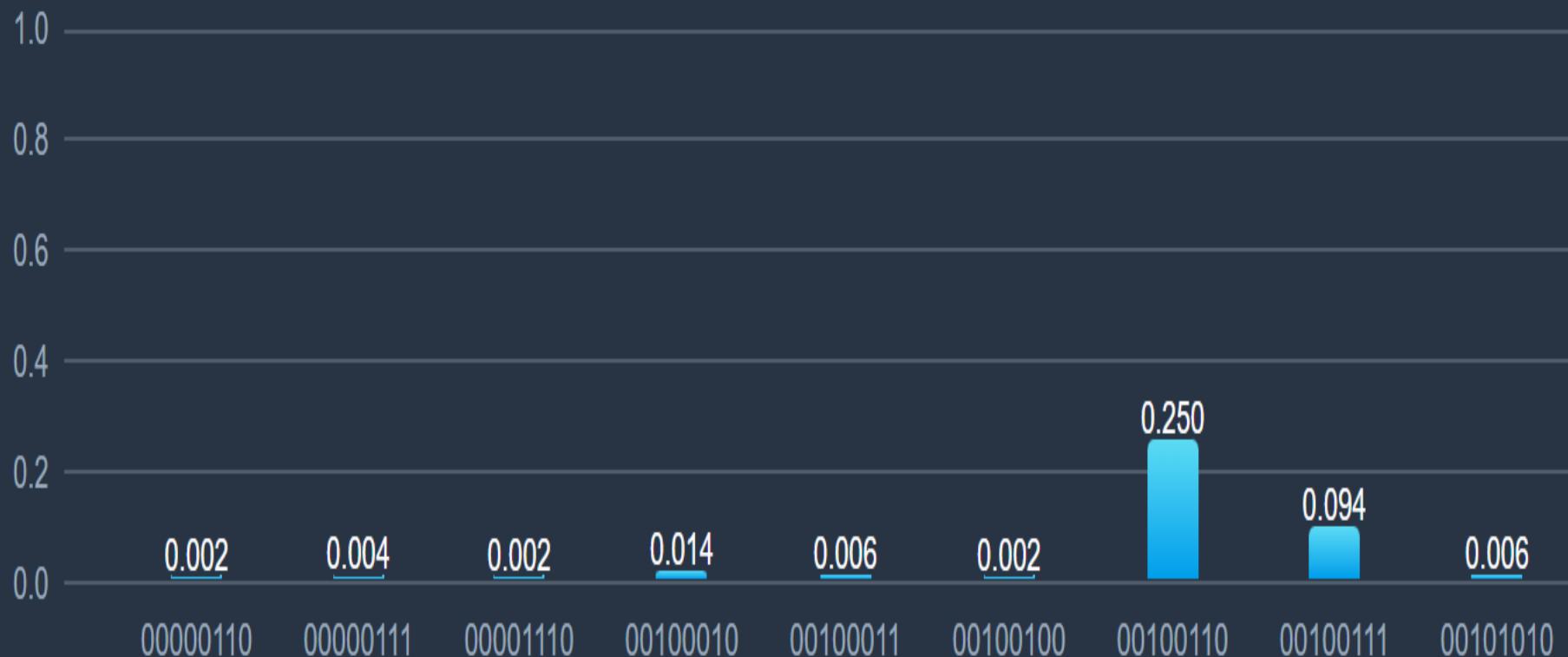
```
1 version 1.0
2
3 qubits 3
4 H q[0]
5 # start writing your code here
6 measure q[0:2]
```



# Przykład układu kwantowego



# Wyniki



Editor Results

```
1 version 1.0
2
3 qubits 8 #the size of produced part of particle
4 # Quantum improvemt for population Q(t)
5 # Virtual machine V is migrated to the host determined by this code
6 # prep_z q[0:3] #state initialization in the z-basis in the |0> state
7 # prep_y q[0:3] #state initialization in the y-basis |R> state
8 # prep_x q[0:3] # state initialization in the x-basis |+>
9 H q[0:7] #execute Hadamard gate on all qubits
10 Rx q[0],-1.23 # rotation of -1.23 radians on qubit 0
11 Rx q[1],1.72
12 Rx q[2],-0.77
```

# Symulator Quantum Inspire

<https://www.quantum-inspire.com/>

version 1.0

```
qubits 8 #the size of produced part of particle
# Quantum improvment for population Q(t)
# Virtual machine V is migrated to the host determined by this code
# prep_z q[0:7] #state initialization in the z-basis in the |0> state
# prep_y q[0:7] #state initialization in the y-basis |R> state
# prep_x q[0:7] # state initialization in the x-basis |+> state
H q[0:7] #execute Hadamard gate on all qubits
```

Rx q[0],-1.23 # rotation of -1.23 radians on qubit 0

Rx q[1],1.72

Rx q[2],-0.77

Rx q[3],-0.27

Rx q[4],-1.24 # rotation of -1.23 radians on qubit 4

Rx q[5],1.62

Rx q[6],-0.70

Rx q[7],-0.37



A tak piłki rotuje Iga Świątek.

```
Ry q[0],-0.50
Ry q[1],1.51 # rotation of 1.51 radians around the y-axis on qubit 1
Ry q[2],1.20
Ry q[3],-0.55
Ry q[4],-1.21 # rotation of -1.21 radians on qubit 4
Ry q[5],1.79
Ry q[6],-0.47
Ry q[7],-0.26
```

```
measure q[0:7] # reading the correction
display_binary # writes the current state of the
measurement register or part of the measurement
register to the simulator output file
```

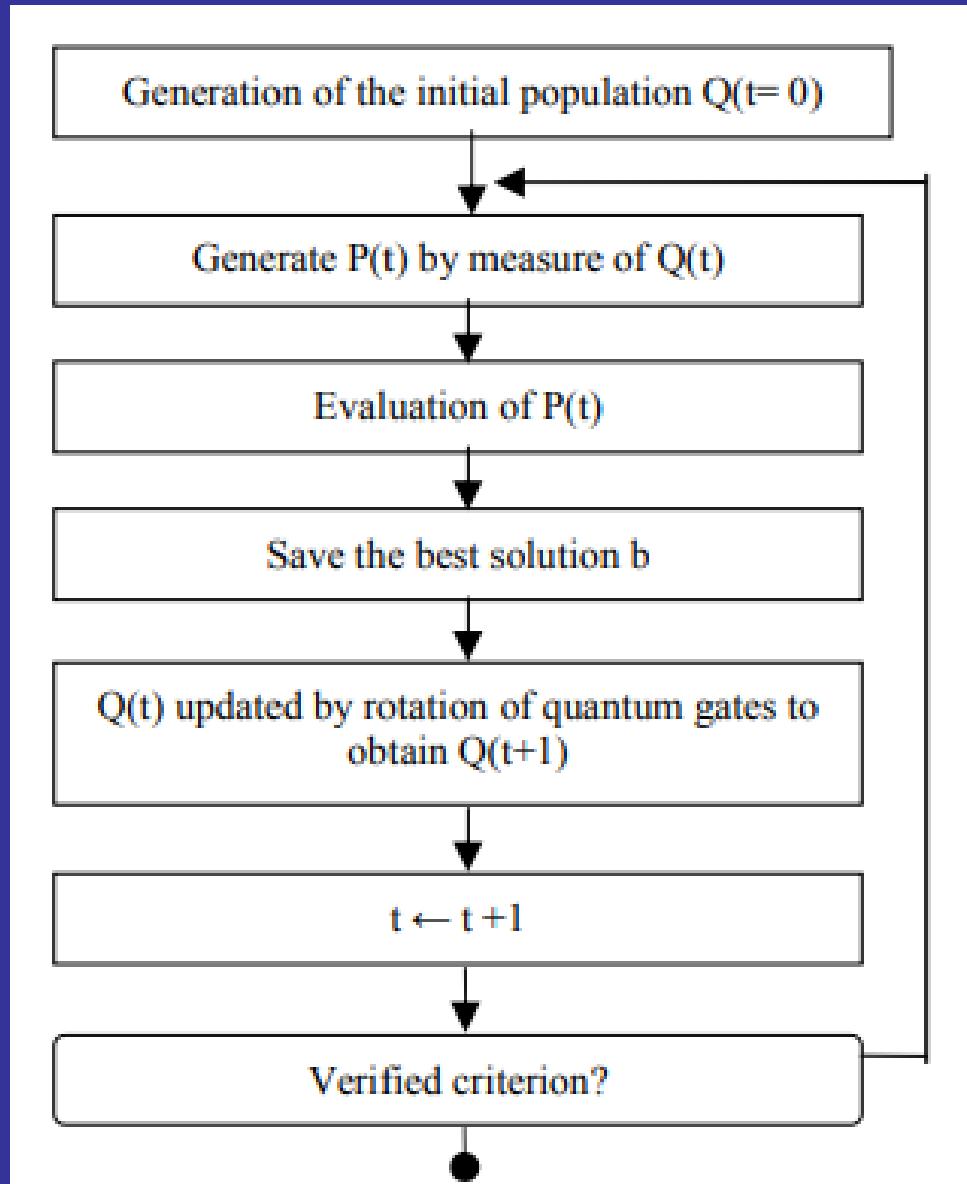
# **Algorytm genetyczny (GA)**

**GA to algorytm przeszukujący losowo za pomocą iteracyjnej transformacji populacji matematycznych obiektów (zazwyczaj liczb binarnych o stałej długości, a każdy obiekt ma wartość przystosowania) w nową populację potomnych obiektów, używając zasad selekcji naturalnej oraz operacji genetycznych takich, jak krzyżowanie i mutacja.**

# Kwantowy Algorytm Genetyczny działa na rejestrze kwantowym $Q$

$$|\Psi\rangle = \alpha |0\rangle + \beta |1\rangle$$

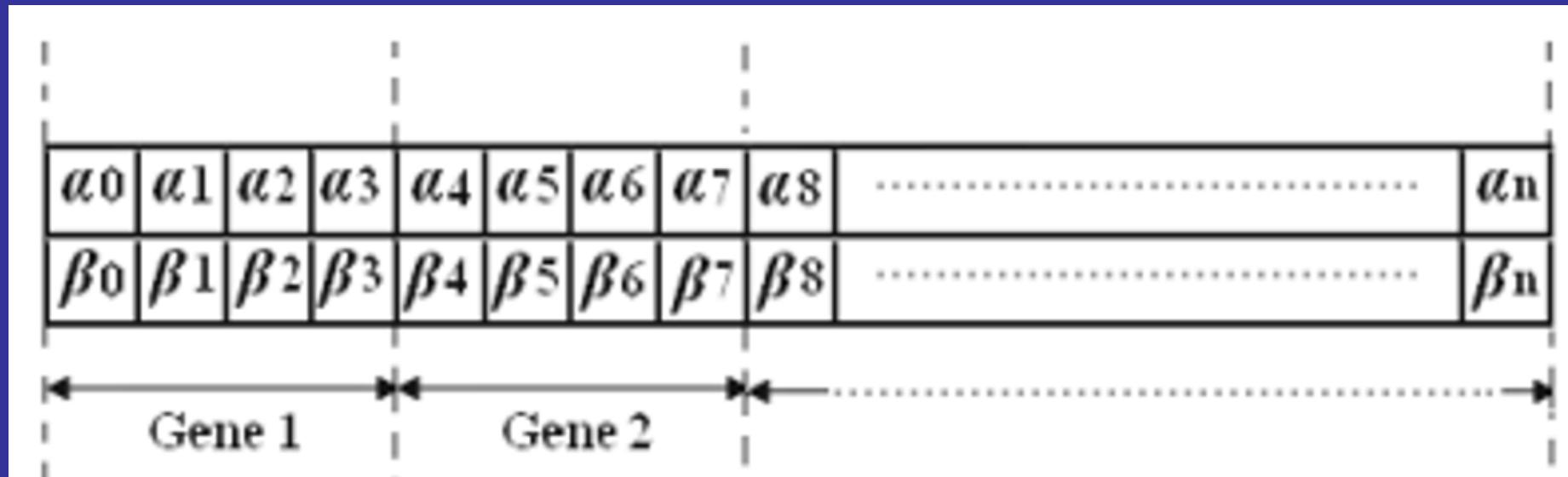
$$|\alpha|^2 + |\beta|^2 = 1$$



# Generowanie cyfrowej populacji



# Kodowanie chromosomu



$$|\Psi\rangle = \alpha |0\rangle + \beta |1\rangle$$

$$|\alpha|^2 + |\beta|^2 = 1$$

# Problem plecakowy

$$f(x) = \sum_{i=1}^m p_i x_i$$

Subject to:

$$f(x) = \sum_{i=1}^m w_i x_i \leq C$$

# Parametry bramek rotacyjnych

xi	bi	f(x) > f(b)	$\Delta\theta_i$	s (ai bi)			
				ai.bi > 0	ai.bi < 0	ai = 0	bi = 0
0	0	0	0.001π	-	+	±	±
0	0	1	0.001π	-	+	±	±
0	1	0	0.08π	-	+	±	±
0	1	1	0.001π	-	+	±	±
1	0	0	0.08π	+	-	±	±
1	0	1	0.001π	+	-	±	±
1	1	0	0.001π	+	-	±	±
1	1	1	0.001π	+	-	±	±

xi to i-ty bit rozwiæzania x ,

bi to i-ty bit najlepszego rozwiæzania b,

f – fitness,

S (ai bi) to znak kąta rotacji  $\theta_i$ .

# Uwagi o rejestrze kwantowym w Quantum Inspire

## **qubits n # Initialize a n-qubit register**

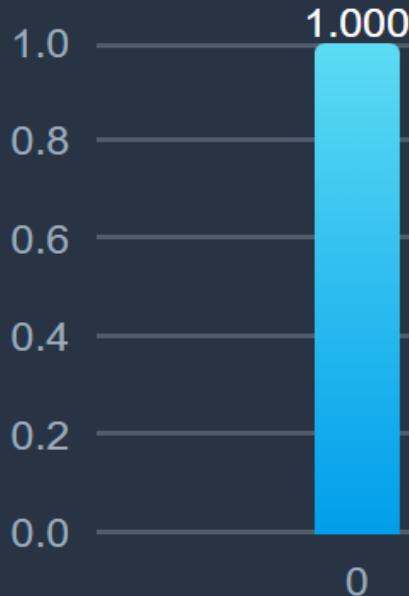
Polecenie **qubits n** inicjalizuje rejestr kubitowy o rozmiarze **n**. Domyślnie wszystkie kubity są inicjowane w stanie  $|0\rangle$  (zera kwantowego).

Maksymalny rozmiar rejestrów kubitów zależy od wybranego backendu do symulacji i ograniczeń konta użytkownika – zazwyczaj 34 kubity

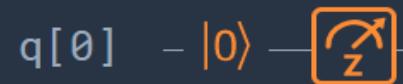
# Inicjowanie rejestru kwantowego

- Całkowita pamięć przydzielona do algorytmu jest określona przez rozmiar rejestru kubitów.
- Nie przydziela się rejestru kubitów większego niż liczba kubitów użytych w algorytmie.
- Minimalizuje to zużycie pamięci i czas wykonania algorytmu.
- Wraz z rejestrem kubitowym zostanie utworzony **rejestr binarny** mający taką samą liczbę bitów jak rejestr kubitowy i służący do przechowywania wyników pomiarów.

# Deterministyczne ustawienie zera kwantowego



```
1 version 1.0
2
3 qubits 1
4 prep_z q[0]
5 # start writing your code here
6 measure q[0]
```



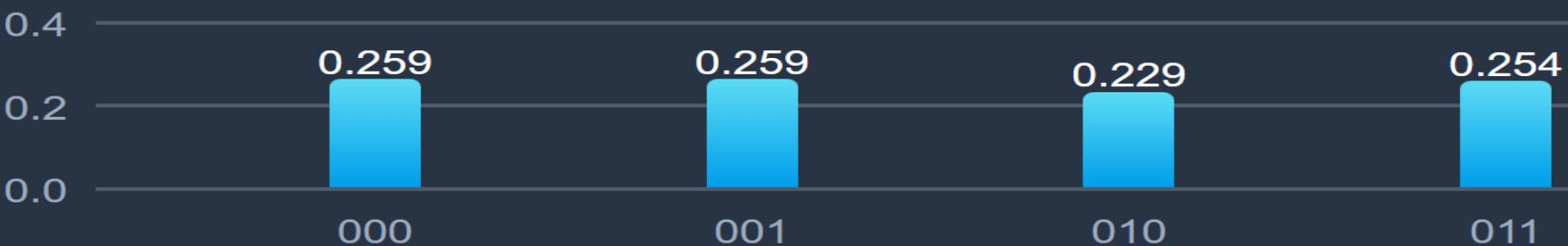
# Punkty charakterystyczne kubitów

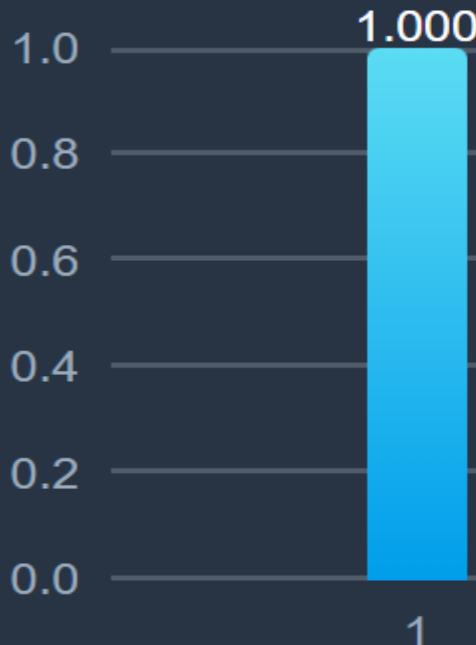
```
1 version 1.0
2
3 qubits 3
4 prep_x q[0]
5 prep_y q[1]
6 prep_z q[2]
7 measure q[0:2]
```

q[0] -  $|+\rangle$

q[1] —  $|R\rangle$

q[2] —  $|0\rangle$





Bramka Pauli-X oznaczana jako X to jednokubitowa rotacja przez  $\pi$  radianów wokół osi x.

```
1 version 1.0
2
3 qubits 1
4 X q[0]
5
6
7 # start writing your code here
```

q[0]

0.512

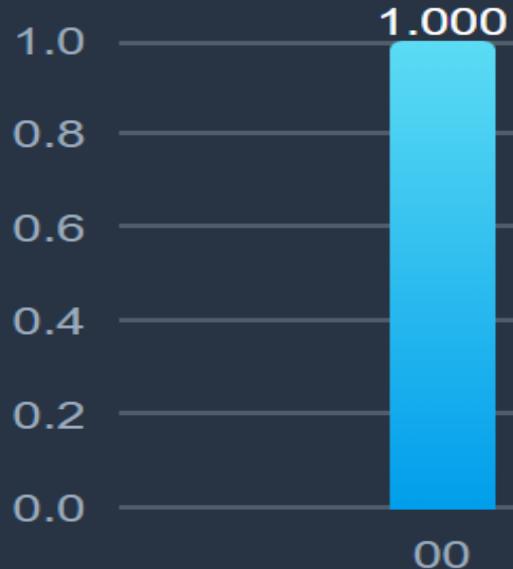
0.488

Stosujemy bramkę Pauliego-X.  
Jeśli kubit był w stanie  
superpozycji po wykonaniu  
operatora Hadamarda, to w nim  
pozostanie.

```
1 version 1.0
2 qubits 1
3 # Inicjalizacja kubitów w stanach #
4 H q[0]
5 X q[0]
```



# Splątanie stanów

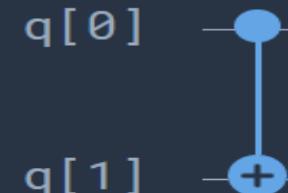


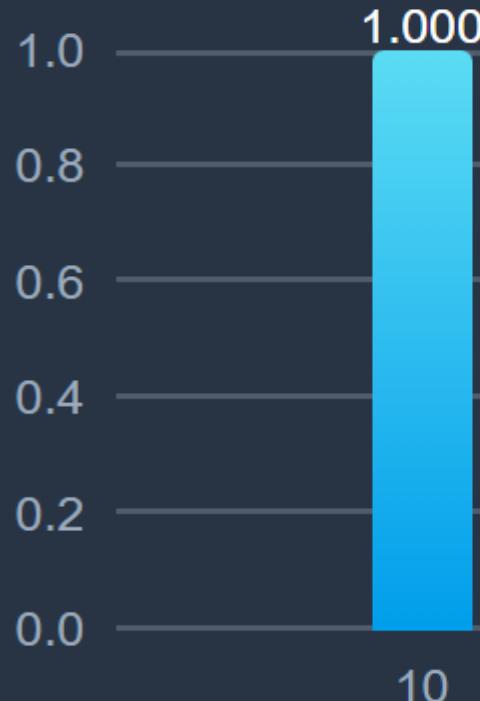
Bramka CNOT to operacja na dwóch kubitach, w której pierwszy kubit jest kubitem kontrolującym, a drugi kubitem zmienianym.

Bramka CNOT pozostawia kubit kontrolujący bez zmian i wykonuje bramkę Pauli-X na kubicie zmienianym, jeśli kubit kontrolujący jest w stanie  $|1\rangle$ ;

CNOT pozostawia kubit docelowy niezmieniony, ponieważ kubit kontrolujący jest w stanie  $|0\rangle$ .

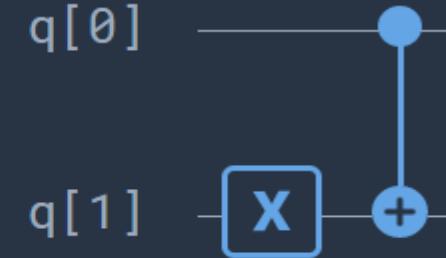
```
1 version 1.0
2
3 qubits 2
4 CNOT q[0],q[1] # CNOT gate between qubits
5 # start writing your code here
```

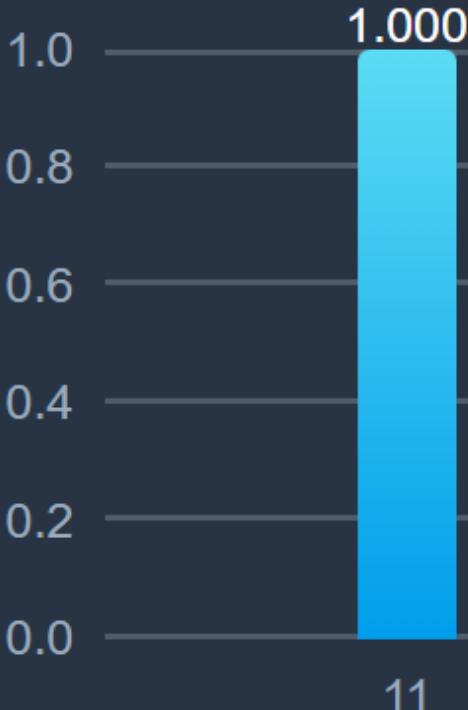




Jeśli kubit kontrolujący  $q[0]$  jest w stanie  $|0\rangle$ , to nie zachodzi negacja kubitu docelowego;  
Jeśli zatem kubit docelowy  $q[1]$  był w stanie  $|1\rangle$ , to również pozostanie w tym stanie.

```
1 version 1.0
2 qubits 2
3 X q[1]
4 CNOT q[0],q[1] # CNOT gate between qubits 0 and 1
5 # start writing your code here
```



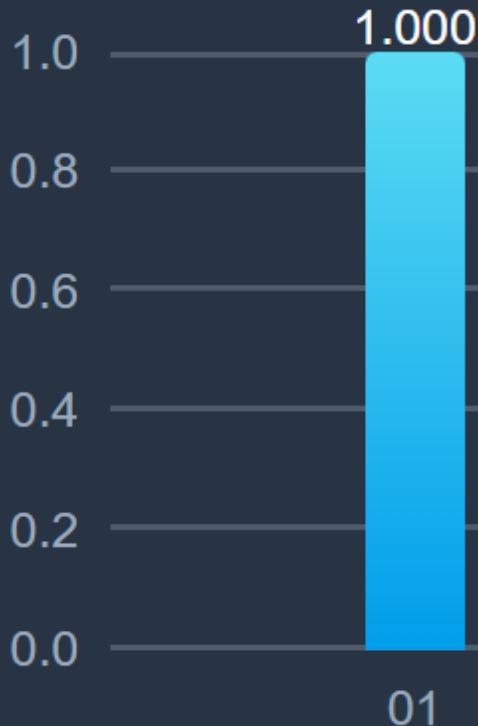


Bramka CNOT wykonuje bramkę Pauli-X na docelowym kubicie, jeśli kubit kontrolujący jest w stanie  $|1\rangle$ ;

W tym wypadku następuje zmiana kubitu docelowego  $q[1]$  z  $|0\rangle$  na  $|1\rangle$ , ponieważ kubit kontrolujący jest w  $|1\rangle$ ;

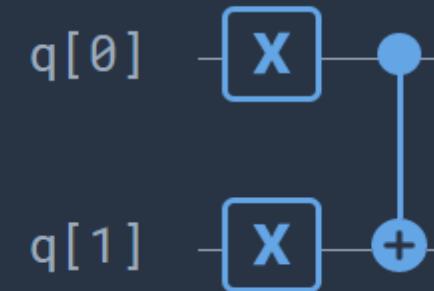
```
1 version 1.0
2 qubits 2
3 X q[0]
4 CNOT q[0],q[1] # CNOT gate between qubits 0 and 1
5 # start writing your code here
```





Bramka CNOT wykonuje bramkę Pauli-X na docelowym kubicie, jeśli kubit kontrolujący jest w stanie  $|1\rangle$ ;  
W tym wypadku następuje zmiana kubitu kontrolowanego  $q[1]$  z  $|1\rangle$  na  $|0\rangle$ .

```
1 version 1.0
2 qubits 2
3 X q[0:1]
4 CNOT q[0],q[1] # CNOT gate between qubits 0 and 1
5 # start writing your code here
```



# Problem decyzyjny, czy funkcja jest stała czy zbalansowana

- Niech  $f : \{0, 1\}^n \rightarrow \{0, 1\}$
- $n$  – liczba bitów;
- Algorytm powinien orzec (wyrocznia, czarna skrzynka), czy funkcja  $f$  jest stała czy też zbalansowana (zwraca 0 dla połowy dziedziny i 1 dla drugiej połowy) – nie może być innej opcji;
- Algorytm deterministyczny wymaga  $2^{n-1} + 1$  ewaluacji funkcji  $f$ ;
- Algorytm probabilistyczny wymaga wykonanie relatywnie dużej liczby mewaluacji, aby prawdopodobieństwo poprawnej decyzji było nie mniejsze niż 0,5<sup>(m-1)</sup>

# Pierwsza historyczna supremacja

## Algorytm Deutscha-Jozsy

- Pierwszy algorytm kwantowy opracowany przez Dawida Deutscha i Richarda Jozsę w 1992;
- Algorytm jest wykładowczo szybszy od każdego możliwego deterministycznego, klasycznego algorytmu;
- Zawsze zwraca poprawną odpowiedź;
- Wyjście:  
00 – funkcja stała; 01 – funkcja zbalansowana;

```

version 1.0
qubits 2
# Inicjalizacja kubitów w stanach
# |+> oraz |->
.initialize
prep_z q[0:1]
X q[1]
{H q[0]|H q[1]}

.oracle_fc1
# do nothing or I q[0:1]
#.oracle_fc2
# X q[1]
#.oracle_fb3
# CNOT q[0],q[1]
#.oracle_fb4
# CNOT q[0],q[1]
# X q[1]
# funkcja stała f(x)=fc1=0 lub
# f(x)=fc2=1
# f. zbalansowana f(x)=fb3=x
# lub f(x)=fb4=NOT(x)

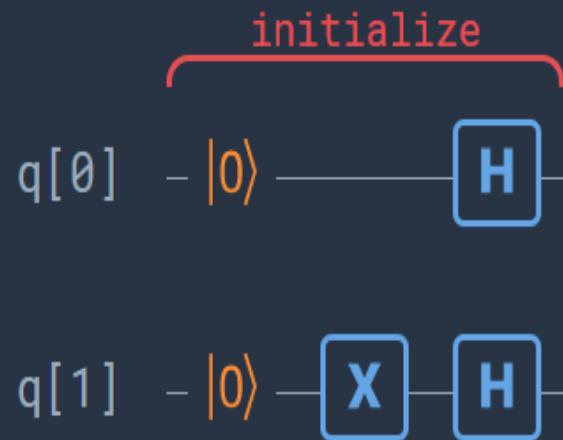
    1.  $F_1(x) = 0$ 
    2.  $F_2(x) = 1$ 
    3.  $F_3(x) = x$ 
    4.  $F_4(x) = 1 - x$ 

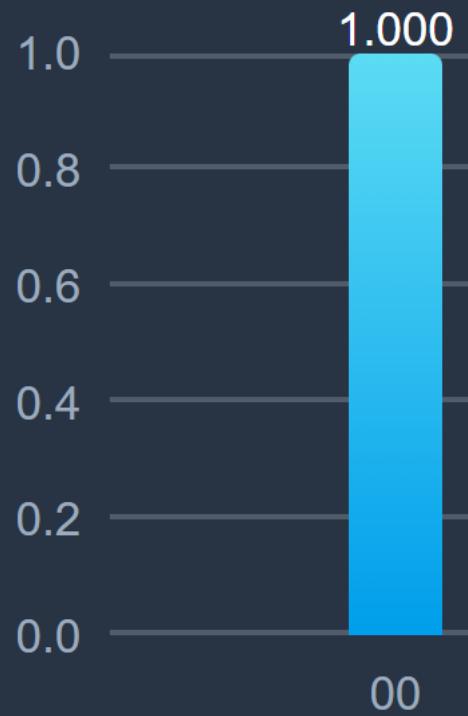
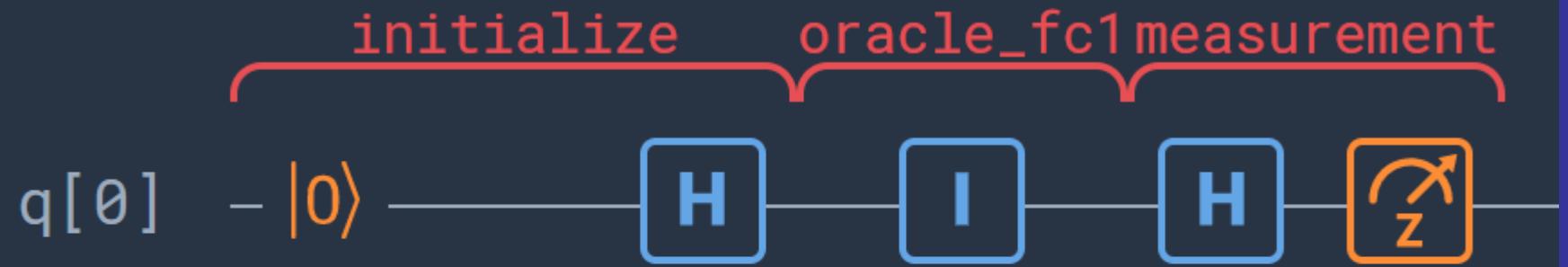
# Bada się tylko jeden przypadek
# funkcji f(x) oracle
# Odkomentowując odpowiednią
# procedurę wyroczni, 4 różne
# przypadki funkcji będą
# testowane.
.measurement
H q[0]
measure q[0]

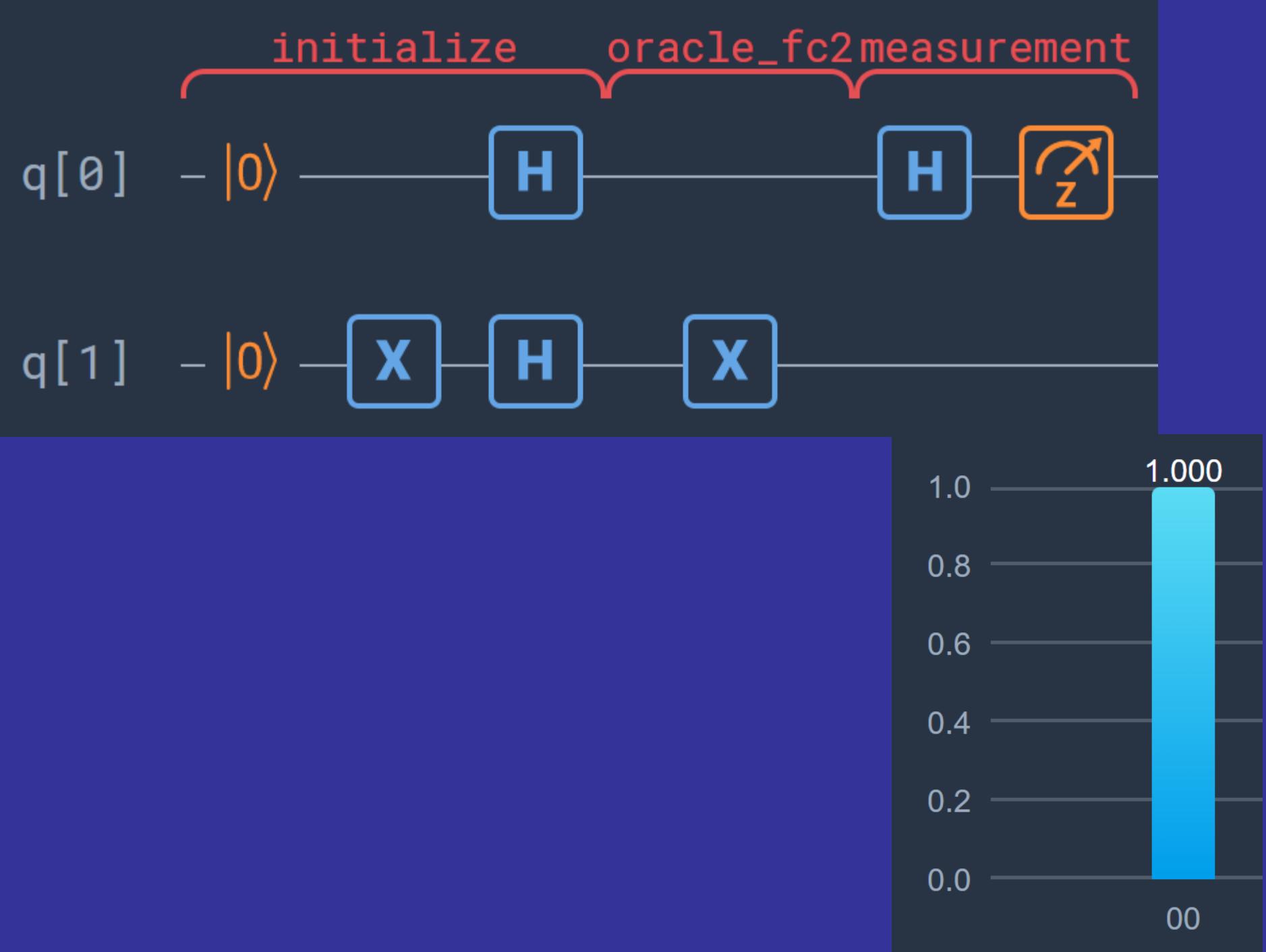
```

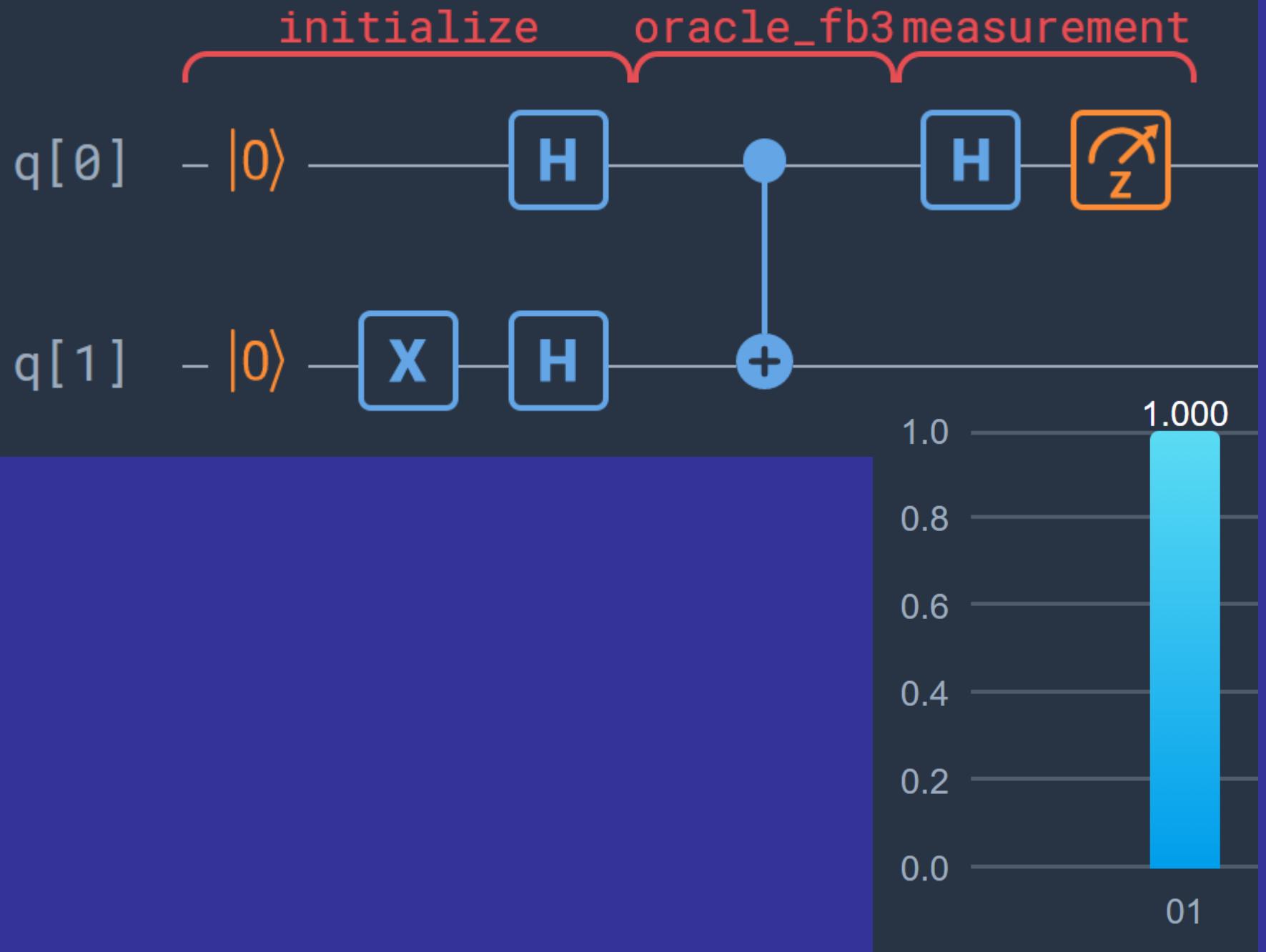


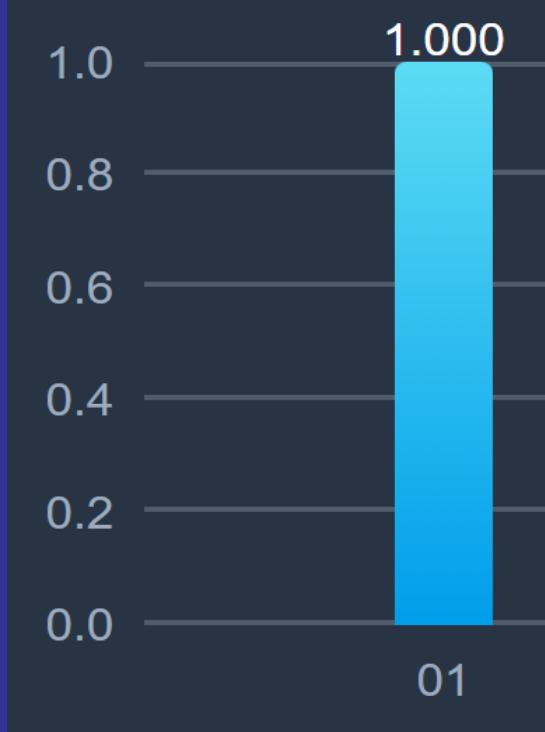
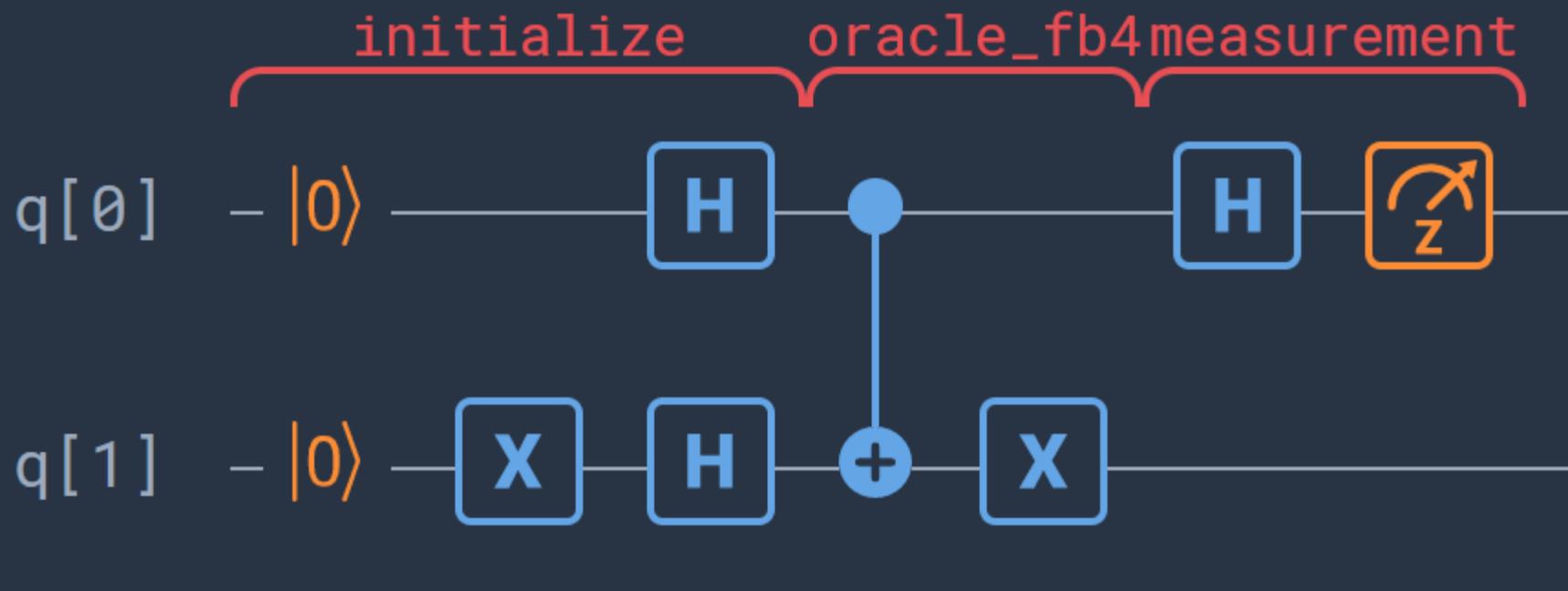
```
1 version 1.0
2 qubits 2
3 # Inicjalizacja kubitów w stanach #
4 .initialize
5 prep_z q[0:1]
6 X q[1]
7 {H q[0] | H q[1]}
```











# Literatura

1. Jerzy Balicki: Many-Objective Quantum-Inspired Particle Swarm Optimization Algorithm for Placement of Virtual Machines in Smart Computing Cloud. Entropy 2022, 24, 58. <https://doi.org/10.3390/e24010058>
2. Edward Farhi, Hartmut Neven: Classification with Quantum Neural Networks on Near Term Processors, 2018.
3. Sanjay Gupta and RKP Zia. Quantum neural networks. Journal of Computer and System Sciences, 63(3):355–383, 2001.
4. Michael A. Nielsen and Isaac L. Chuang. Quantum computation and quantum information. Cambridge University Press, 2010.
5. Jerzy Tchórzewski, Dariusz Ruciński: Kwantowa sztuczna sieć neuronowa część 1. Metoda i wyniki obliczeń, 2018.

