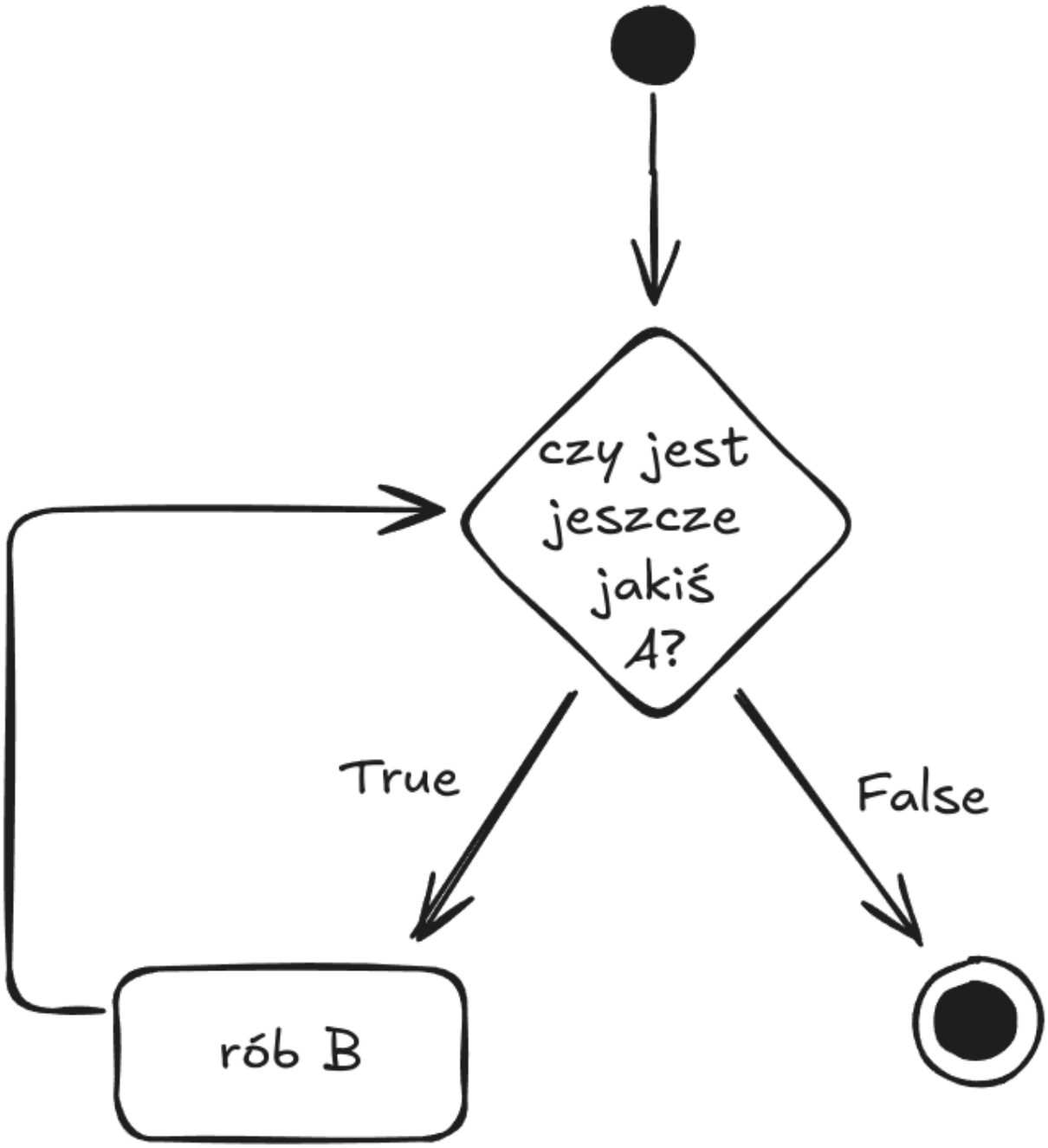
# P?tle, modu? ?ó?wia

Do wykonywania powtarzalnych czynności używamy w programowaniu pętli.

# P?tla for

Wykonuje blok kodu tyle razy ile mamy jakiś elementów w sekwencji (np. tyle razy ile mamy liter w wyrazie, elementów w liście) / pozostałych obiektach iterowalnych




np.

```
for litera in "Robisz.to":
    print(litera)
```

```
for element in ["a", "b", "c"]:
    print(element)
```

Na początku najważniejsze (i prostsze do zrozumienia) jest przyjęcie, że obiektem dla pętli for są sekwencje / inne iterowalne elementy - ciągi znaków, listy, krotki, range.

Nie ma obowiązku wykorzystywania elementu sekwencji w powtarzalnym bloku. Jeśli używamy go jedynie do przemieszczania się po sekwencji to dobrą praktyką jest nazwanie go "`_`" (czyli użycie znaku podłogi).

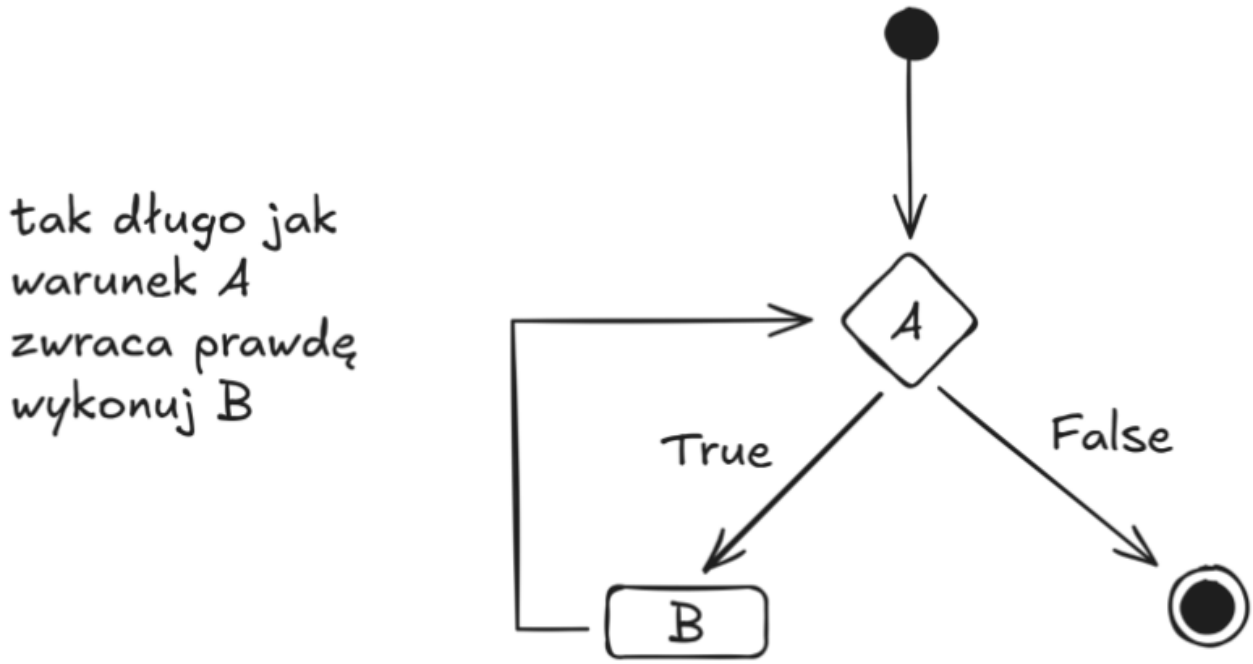
Ogólny schemat korzystania z tej pętli:


*Na powyższym obrazku słowo "sekwencja" jest użyte w potocznym rozumieniu. Tak jak wspomniano wyżej chodzi o każdy iterowalny element. Na start można jednak nie zawracać sobie głowy tymi subtelnościami - dla zainterowanych link w odnośnikach.*

# P?tla while

"... wykonuj tak długo aż"

- pętla ta wykonuje się tak długo jak jakiś warunek jest spełniony




np.

```
i = 0


while i < 10:
    print(i)
    i += 1
```

*Przy każdym "obrocie" pętli zwiększamy "i" o 1, gdy osiągnie 10 warunek "i < 10" zwróci* `False` *i pętla przestanie się wykonywać.*

# Break

Jeśli chcemy wyjść z pętli wcześniej (w przypadku `for` zanim dojdziemy do ostatniego elementu, w przypadku `while` zanim warunek zwróci `False`) możemy skorzystać z `break`.

```
i = 0


while True:
    if i < 10:
        break
    print(i)
```

```
    i += 1
```

# Funkcja range

Jeśli nie mamy jakiejś gotowej sekwencji z odpowiednią ilością elementów to z pomocą może przyjść nam funkcja range.

W najprostszej postaci konstruujemy ją tak:

**range(ilośc_elementów)**

np. chcąc wyświetlić napis "Hej" 3 razy możemy zrobić to tak:

```
for _ in range(3):
    print(Hej)
```

Jeśli jednak potrzebujemy jakiegoś konkretnego zakresu (np. od 10 do 20 etc) i o konkretnym kroku (czyli elementy zwiększają się o konkretną wartość) można skorzystać z dodatkowych parametrów:

**range(start, stop, krok)**

# Importowanie bibliotek

Jeśli problem, który staramy się rozwiązać nie jest trywialny dobrym pomysłem może być wykorzystanie kodu napisanego przez kogoś innego.

Python dysponuje szeroką gamą bibliotek dostępnych za darmo. Całkiem pokaźna ich ilość jest instalowana razem z Pythonem.

Jeśli potrzebujemy jakiejś zewnętrznej (np. znalezionej na www.pypi.org ) możemy ją zainstalować poprzez wykonanie w terminalu:

```
pip3 install nazwa_bibilioteki
```

Aby wykorzystać zainstalowaną bibliotekę w pliku z programem nad którym się pracuje należy na początku zaimportować ją poprzez:

**import nazwa_biblioteki**

np.

```
import turtle
```

Jeśli jest potrzeba zaktualizowania biblioteki do najnowszej wersji można:

```
pip3 install -U <NAZWA_BIBLIOTEKI>
```

# Modu? Turtle

Służy do rysowania z pomocą kodu. Idealnie nadaje się do ćwiczeń pętli - od razu widać co się dzieje, tym samym łatwiej diagnozować błędy.

Importuje się go poprzez:

```
import turtle
```

# Przygotowanie programu dla "?ó?wia"

Po zaimportowaniu biblioteki należy stworzyć "żółwia" i przestrzeń po której będzie się poruszał:

`zolwik =  turtle.Turtle`

Z kolei pod koniec naszego kodu (po tym jak już skończymy rysować) powinniśmy dodać:

`turtle.exitonclick()`

- to spowoduje, że okno z programem nie zamknie się natychmiast po tym jak żółwik skończy pracę

# Podstawowe metody

Wykonujemy je na obiekcie Turtle (ja zazwyczaj nazywam go zolwik).

**.forward(odległość)** - przesuwa żółwia do przodu

**.left(kąt)** - żółwik skręca w lewo o podany kąt

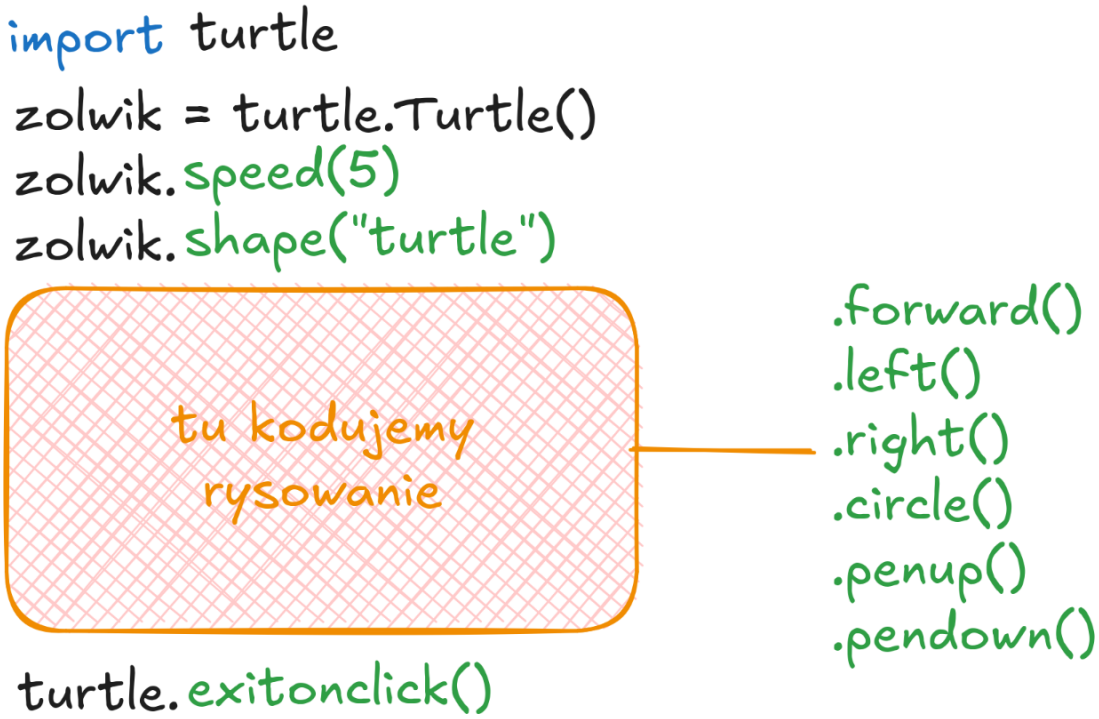
**.right(kąt)** - skręca w prawo o podany kąt

**.circle(promień)** - rysuje okrąg o podanym promieniu

**.penup()** - żółwik przemieszczając się przestanie zostawiać ślad

**.pendown()** - ponownie zacznie rysować

# Turtle - struktura programu




# Odno?niki

„3.12.5 Documentation”. Dostęp 18 sierpień 2024. https://docs.python.org/3/.

Bunn, Tristan. *Learn Python visually: creative coding with processing.py*. San Francisco, CA: No Starch Press Inc, 2021.

Gruppetta, Stephen. „Iterable: Python’s Stepping Stones”, 22 czerwiec 2024. https://www.thepythoncodingstack.com/p/python-iterable-data-structures.

„Python and Turtle – Python, Turtle, Projects, Learn”. Dostęp 11 sierpień 2024. https://pythonturtle.academy/.

---

Wersja #6
Utworzono 2024-08-26 11:05:10 UTC przez Przemek Jeske
Zaktualizowano 2025-09-22 15:07:17 UTC przez Przemek Jeske