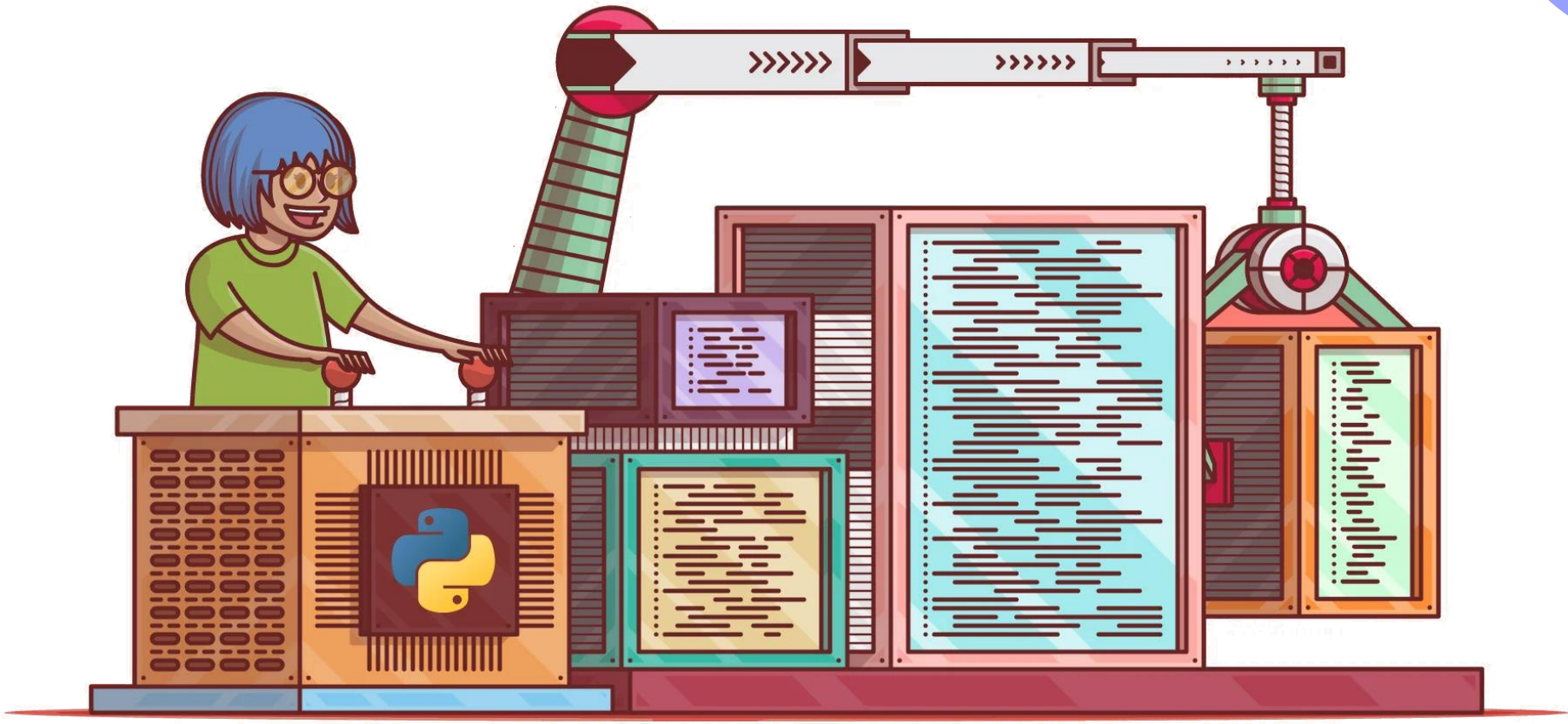
Moduły i pakiety



Real Python

Moduły
(definicja)

Moduł - plik z rozszerzeniem .py

Importowanie modułów

W jednym module możemy zaimportować kod innego modułu. Importowanie modułu oznacza dołączenie kodu znajdującego się w module, który importujemy do kodu modułu do którego importujemy. Służy do tego instrukcja **import**

Importowanie modułów

Możemy importować cały kod modułu

```
import <nazwa_modulu>
```

lub jego fragment

```
from <nazwa_modulu> import <fragment_modulu>
```

Importowanie modułów
(przykład)

```
import math


res = math.floor(3.4)
print(res)  # 3
```

Importowanie modułów
(przykład)

```
from math import floor


res = floor(3.4)
print(res)  # 3
```

Aliasy

Zaimportowanemu kodowi/fragmentom kodu możemy nadawać oddzielne nazwy (aliasy).

```
import <nazwa_modulu> as <nowa_nazwa>
```

```
from <nazwa_modulu> import <fragment_modulu> as <nowa_nazwa>
```

Aliasy
(przykład)

```python
import math as m


res = m.floor(3.4)
print(res)  # 3
```

Aliasy
(przykład)

```python
from math import floor as f


res = f(3.4)
print(res)  # 3
```

Przykłady wbudowanych modułów

(elementów biblioteki standardowej Pythona)

os
sys
time
random
math
unittest
tkinter

W których miejscach Python szuka nazw modułów ?

Podczas wykonywania instrukcji import Python szuka modułu o wskazanej nazwie kolejno w:

- bieżącym katalogu
- wszystkich ścieżkach znajdujących się w zmiennej środowiskowej PYTHONPATH
- katalogu instalacji interpretera Python

Jeżeli w żadnej z powyższych lokalizacji Python nie znajdzie wskazanego modułu to rzuca wyjątek ImportError.

W których miejscach Python szuka nazw modułów ?

Listę ścieżek, które przeszukuje Python podczas szukania wskazanego modułu można sprawdzić za pomocą funkcji **path** z modułu **sys**

```
import sys


print(sys.path)
```

Lokalizacja modułu

Lokalizację modułu na dysku możemy sprawdzić za pomocą atrybutu __file__ tego modułu

```
import random


print(random.__file__)
```

Przestrzeń nazw modułu

Każdy moduł Pythona posiada swoją przestrzeń nazw, którą nazywamy **globalną przestrzenią nazw**. W Pythonie nie istnieje globlana przestrzeń nazw współdzielona przez różne moduły.

Kiedy importujemy moduł, to tak naprawdę importujemy przestrzeń nazw tego modułu do innego modułu.

Pakiety
(definicja)

Pakiet - zbiór modułów (katalog, w którym znajdują się pliki z rozszerzeniem .py).

Marker pakietu

W katalogu, który chcemy żeby był pakietem Pythona należy umiecić plik __init__.py (tzw. marker pythona). Plik __init__.py może być pusty.

Importowanie pakietu

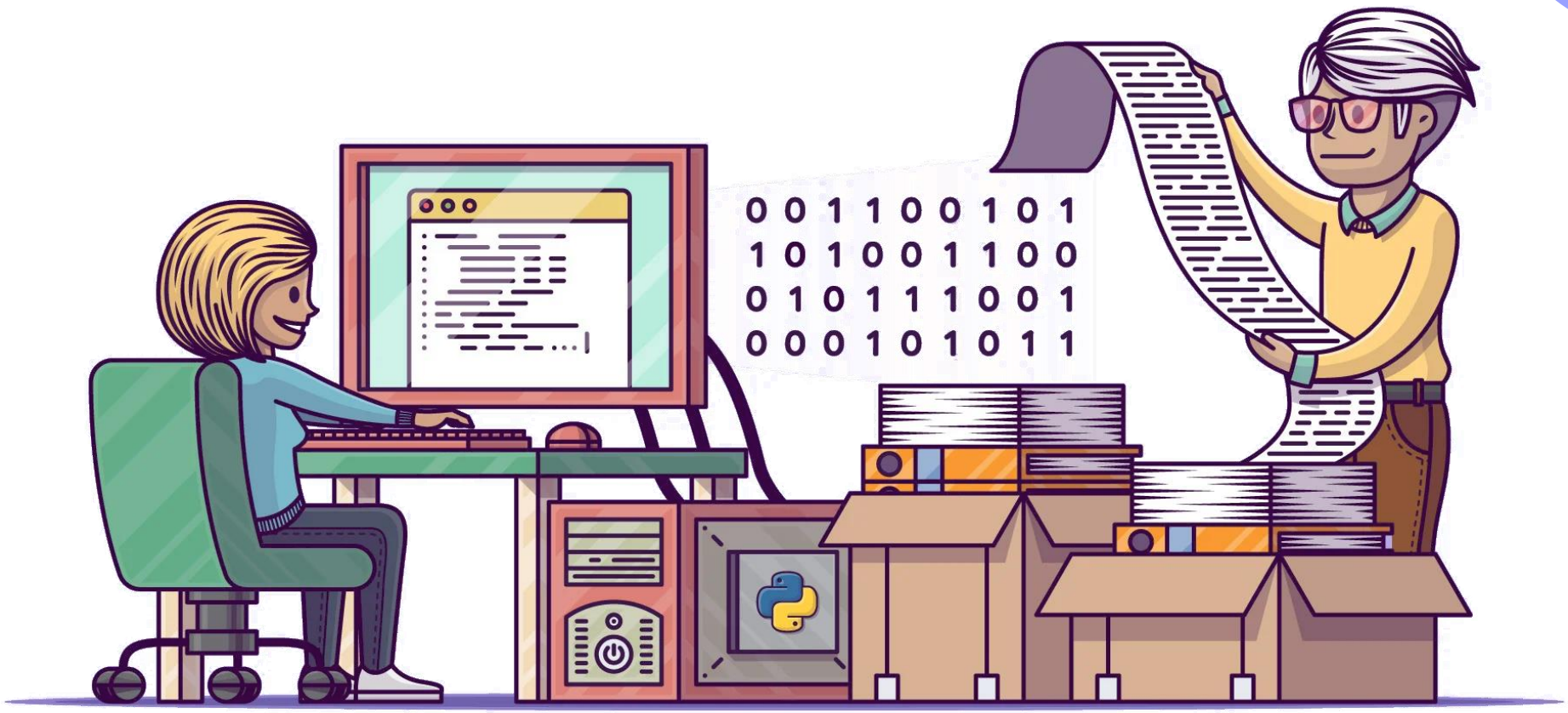
Pakiety importuje w identyczny sposób jak modułu - instrukcją **import**. Tak jak możemy importować fragment modułu, tak możemy też importować wybrany moduł z pakietu - instrukcją **from … import ...**

Menadżer pakietów Python

Poza pakietami znajdującymi się w standardowej bibliotece Pythona istnieją tysiące zewnętrznych pakietów. Do zarządzania zewnętrznymi pakietami służy tzw. menadżer pakietów pythona **pip** (**P**ackage **I**nstaller for **P**ython).

Za pomocą menadżera pip możemy instalować dowolny pakiet pythona, który znajduje się w repozytorium pakietów Pythona, tzw. **PyPI** (**Py**thon **P**ackage **I**ndex)

Pliki



Real Python

Wczytywanie zawartości pliku

Otwarcie pliku w trybie do odczytu

```
f = open("demo.txt", "r")
```

Wczytywanie zawartości pliku

Otwarcie pliku w trybie do odczytu

```
f = open("demo.txt", "r")
```

Wczytanie zawartości pliku

```
content = f.read()
```

Wczytywanie zawartości pliku

Otwarcie pliku w trybie do odczytu

```
f = open("demo.txt", "r")
```

Wczytanie zawartości pliku

```
content = f.read()
```

Zamknięcie pliku

```
f.close()
```

Zapisywanie do pliku

Otwarcie pliku w trybie do zapisu

```
f = open('demo.txt', 'w')
```

Zapisywanie do pliku

Otwarcie pliku w trybie do zapisu

```
f = open('demo.txt', 'w')
```

Zapisanie do pliku

```
f.write("Ala ma kota")
```

Zapisywanie do pliku

Otwarcie pliku w trybie do zapisu

```
f = open('demo.txt', 'w')
```

Zapisanie do pliku

```
f.write("Ala ma kota")
```

Zamknięcie pliku

```
f.close()
```

Tryby otwierania pliku

(drugi parametr funkcji open)

Odczyt:

r – otwarcie pliku w trybie do odczytu (tworzy plik jeżeli taki plik nie istnieje)

Zapis:

a – otwarcie pliku w trybie do dopisywania (tworzy plik jeżeli taki plik nie istnieje)

w – otwarcie pliku w trybie do nadpisywania (tworzy plik jeżeli taki plik nie istnieje)

x – stworzenie pliku (rzuca błąd, jeżeli plik już istnieje)

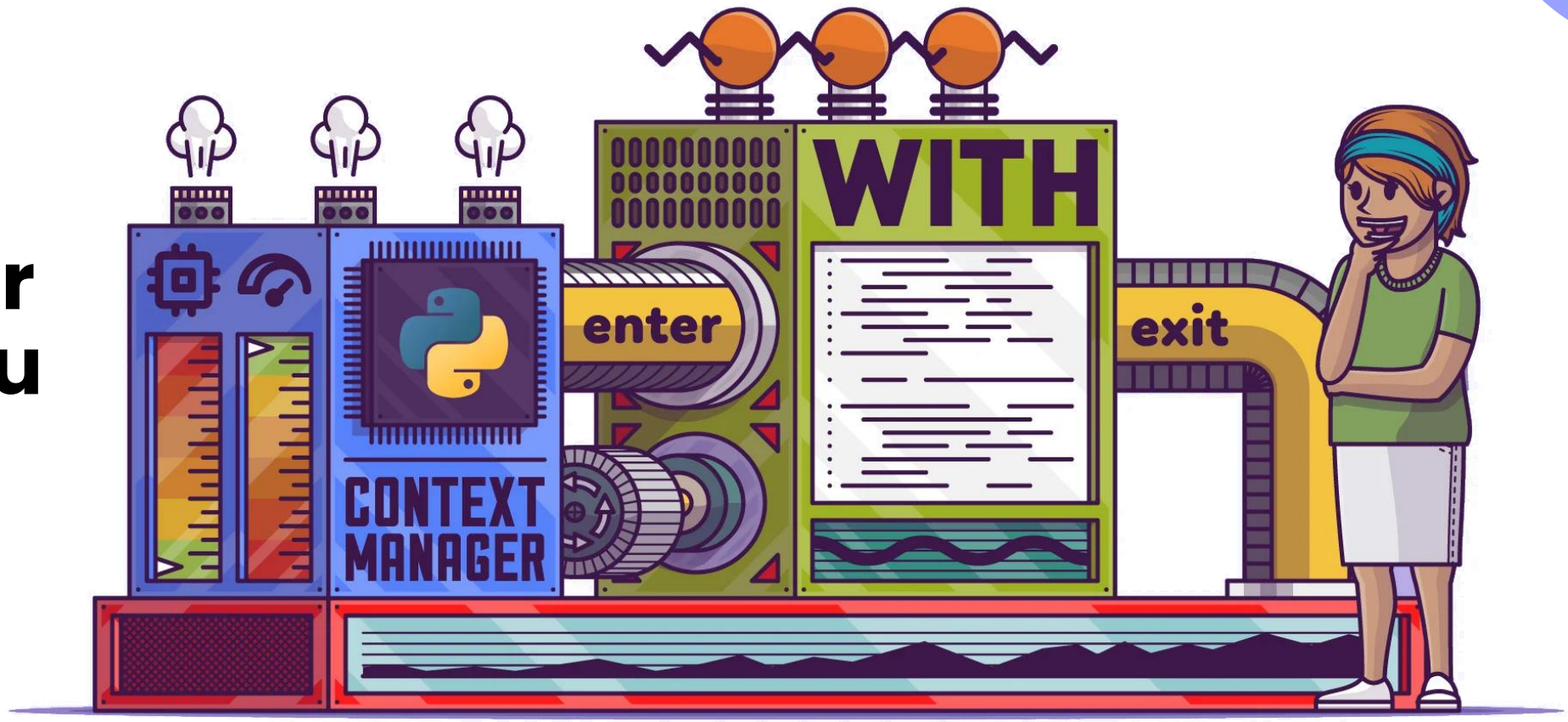
Tryby otwierania pliku

(drugi parametr funkcji open)

t – tryb tekstowy

b – tryb bianarny

Menadżer kontekstu



Wczytywanie zawartości pliku
(menadżer kontekstu)

```
with open('demo.txt', 'r') as f:
    content = f.read()
```

Zapisywanie do pliku
(menadżer kontekstu)

```
with open('demo.txt', 'w') as f:
    content = f.write("Ala ma kota")
```

4 sposoby wczytywania zawartości pliku

1. Wczytanie całej zawartości pliku jako jednego długiego napisu

```
with open('demo.txt', 'r') as f:
    content = f.read()
```

4 sposoby wczytywania zawartości pliku

2. Wczytanie całej zawartości pliku jako listy linijek

```
with open('demo.txt', 'r') as f:
    content_list = f.readlines()
```

4 sposoby wczytywania zawartości pliku

3. Wczytywanie linijka po linijce

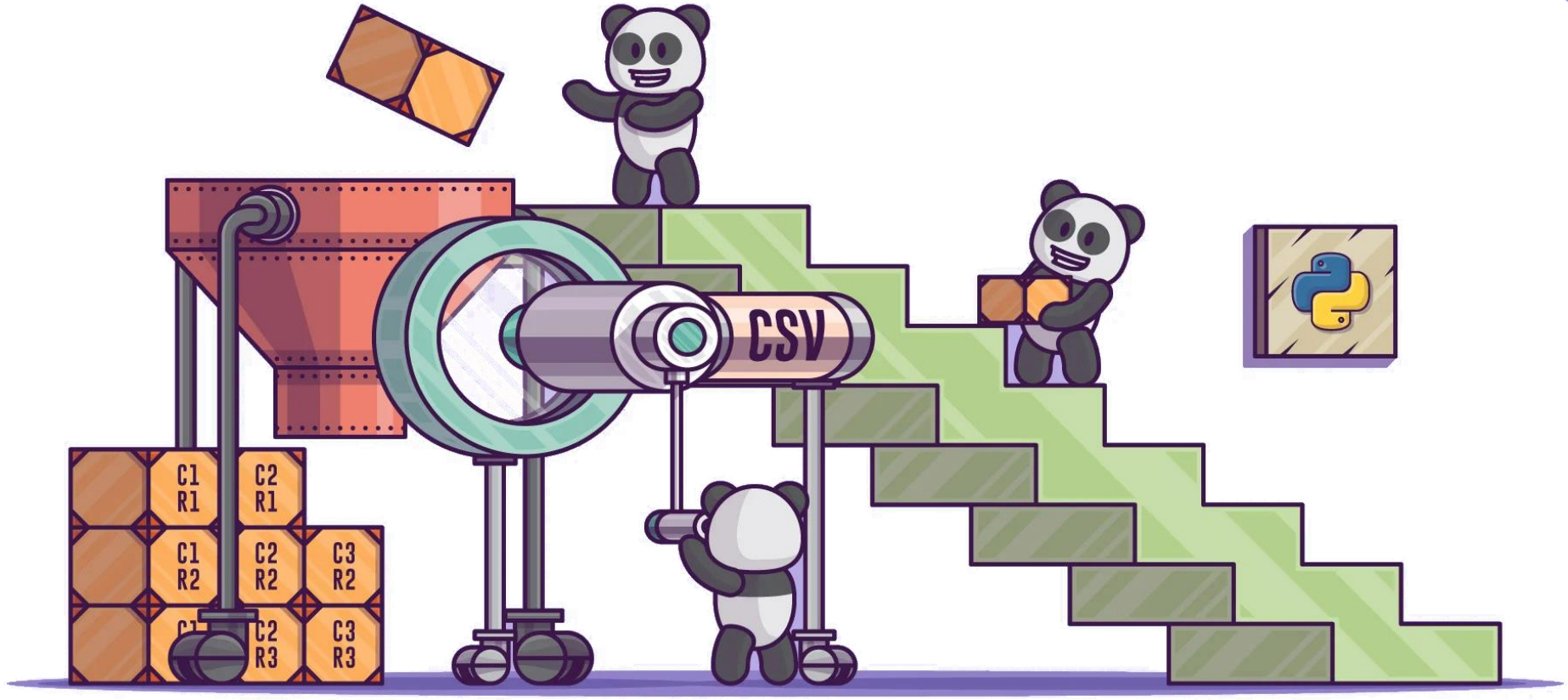
```
with open('demo.txt', 'r') as f:
    while True:
        line = f.readline()

        if not line:
            break
```

4 sposoby wczytywania zawartości pliku

4. Wczytywanie linijka po linijce

```
with open('demo.txt', 'r') as f:
    for line in f:
        print(line)
```

csv



Real Python

csv — CSV File Reading and Writing

Source code: Lib/csv.py

The so-called CSV (Comma Separated Values) format is the most common import and export format for spreadsheets and databases. CSV format was used for many years prior to attempts to describe the format in a standardized way in **RFC 4180**. The lack of a well-defined standard means that subtle differences often exist in the data produced and consumed by different applications. These differences can make it annoying to process CSV files from multiple sources. Still, while the delimiters and quoting characters vary, the overall format is similar enough that it is possible to write a single module which can efficiently manipulate such data, hiding the details of reading and writing the data from the programmer.

The `csv` module implements classes to read and write tabular data in CSV format. It allows programmers to say, "write this data in the format preferred by Excel," or "read data from this file which was generated by Excel," without knowing the precise details of the CSV format used by Excel. Programmers can also describe the CSV formats understood by other applications or define their own special-purpose CSV formats.

The `csv` module's `reader` and `writer` objects read and write sequences. Programmers can also read and write data in dictionary form using the `DictReader` and `DictWriter` classes.

See also:

PEP 305 - CSV File API
The Python Enhancement Proposal which proposed this addition to Python.

Module Contents

Wczytywanie zawartości csv do listy

```
# Reading csv file using csv.reader
import csv

with open('data.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')

    for row in csv_reader:
        print(row)
```

Wczytywanie zawartości csv do słownika

```python
# Reading csv file using csv.DictReader
import csv

with open('data.csv', mode='r') as csv_file:
    csv_reader = csv.DictReader(csv_file)

    for row in csv_reader:
        print(row)
```

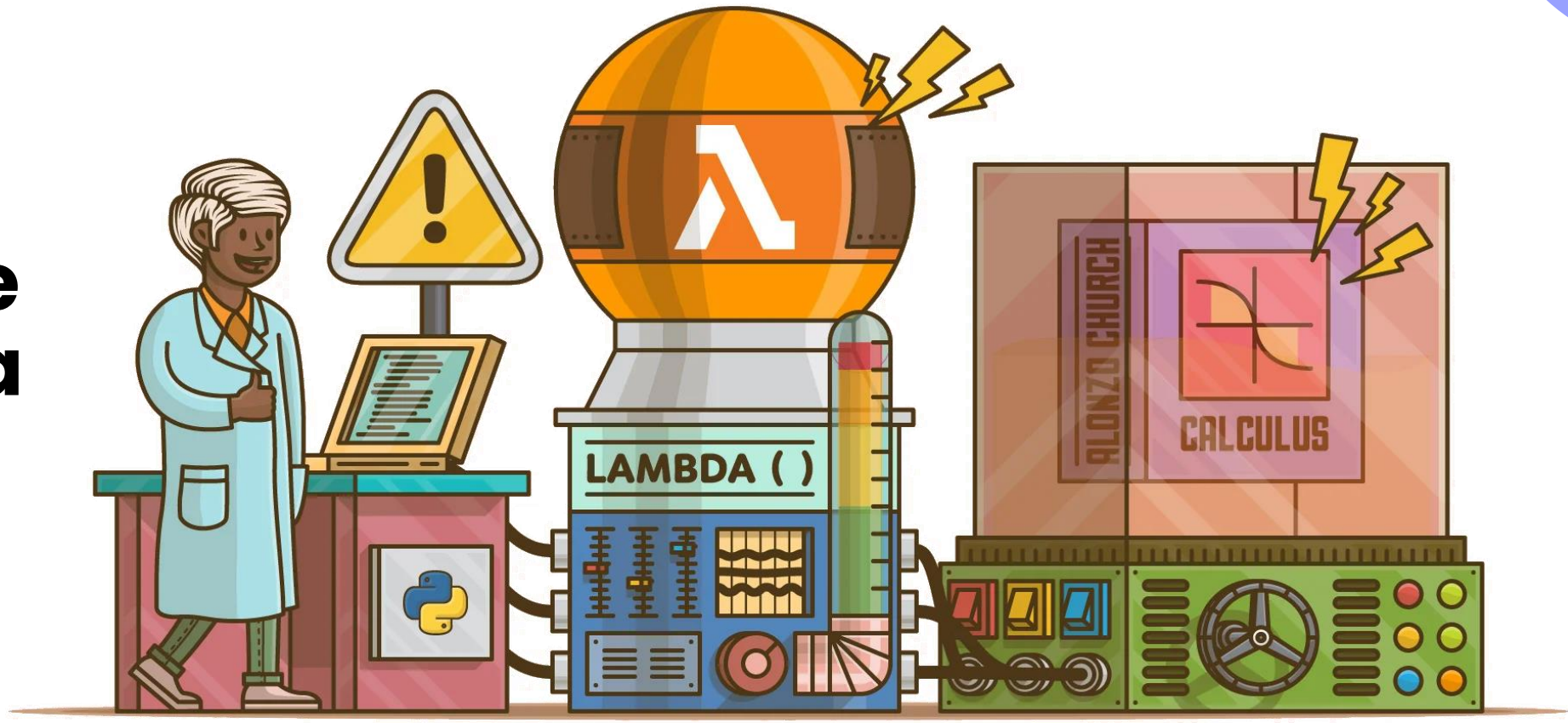
Zapisanie listy do csv

```python
# Writing lists to csv file using csv.writer
import csv

with open('data2.csv', mode='w', newline='') as f:
    employee_writer = csv.writer(f, delimiter=',')

    employee_writer.writerow(['Jan Kowalski', 'HR', 'mar'])
    employee_writer.writerow(['Ewa Nowak', 'IT', 'lis'])
```

Zapisanie słownika do csv

```
# Writing dict to csv file using csv.DictWriter
import csv

with open('data3.csv', mode='w', newline='') as csv_file:
    fieldnames = ['emp_name', 'dept', 'birth_month']
    writer = csv.DictWriter(csv_file, fieldnames=fieldnames)

    writer.writeheader()
    writer.writerow({'emp_name': 'John Smith', 'dept': 'Accounting', 'b
    writer.writerow({'emp_name': 'Erica Meyers', 'dept': 'IT', 'birth_m
```

Funkcje lambda



Real Python

Funkcja lambda

Funkcja lambda – (aka funkcja anonimowa) funkcja nie posiadająca nazwy, definiowana w swoim jedynym miejscu użycia

Funkcje lambda
(przykład)

Zwykła funkcja

```
def square(x):
    return x * x
```

Funkcja lambda

```
square = lambda x: x * x
```

Funkcje lambda
(przykład)

Zwykła funkcja

```
def sum_(x, y):
    return x + y
```

Funkcja lambda

```
sum_ = lambda x, y: x + y
```

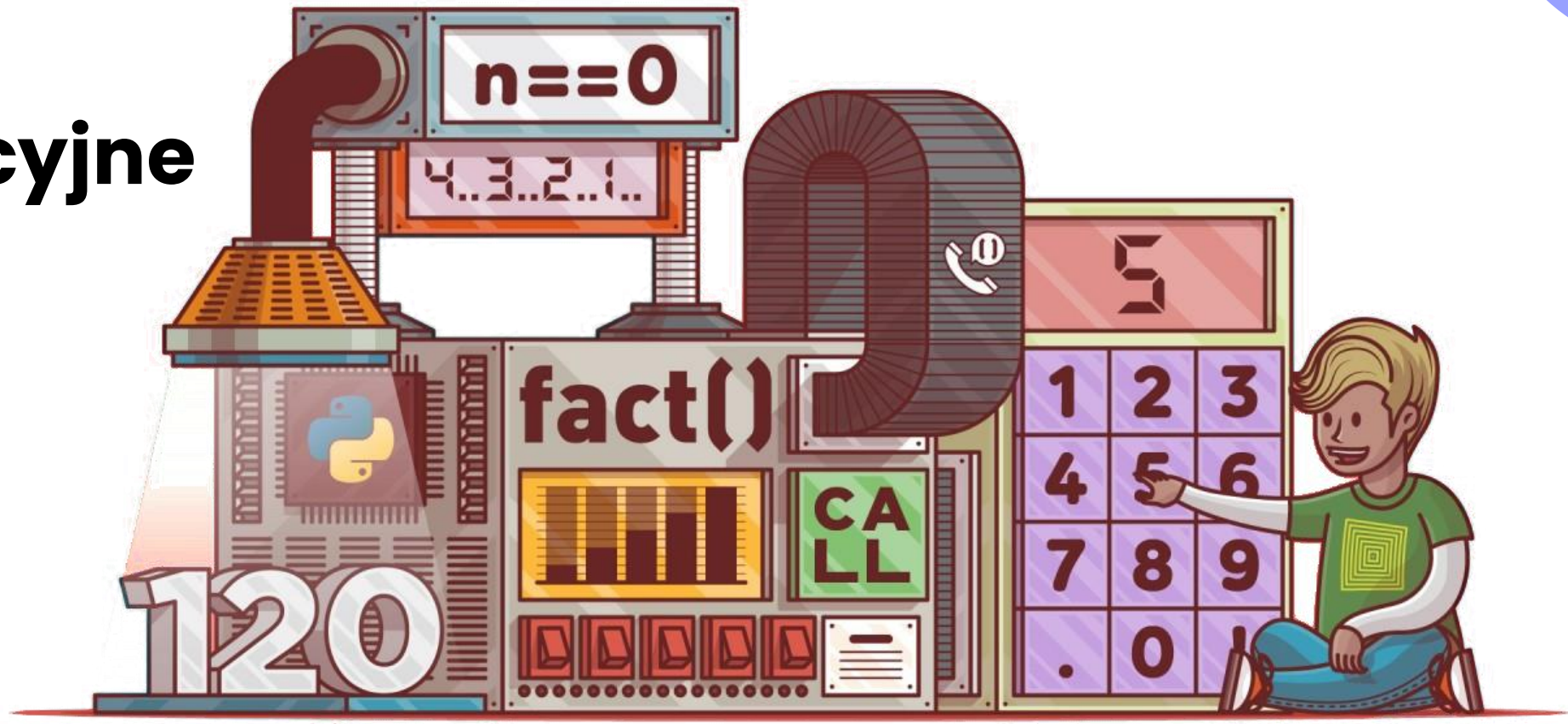
Funkcje lambda
(przykład użycia)

```
pairs = [
    (1, 10),
    (2, 9),
    (3, 8)
]


new *
def get_second_element(data):
    return data[1]

new_pairs = sorted(pairs, key=get_second_element)
print(new_pairs)  # [(3, 8),(2, 9),(1, 10)]
```

Funkcje lambda
(przykład użycia)

```
pairs = [
    (1, 10),
    (2, 9),
    (3, 8)
]

new_pairs = sorted(pairs, key=lambda x: x[1])
print(new_pairs)  # [(3, 8),(2, 9),(1, 10)]
```

Funkcje rekurencyjne

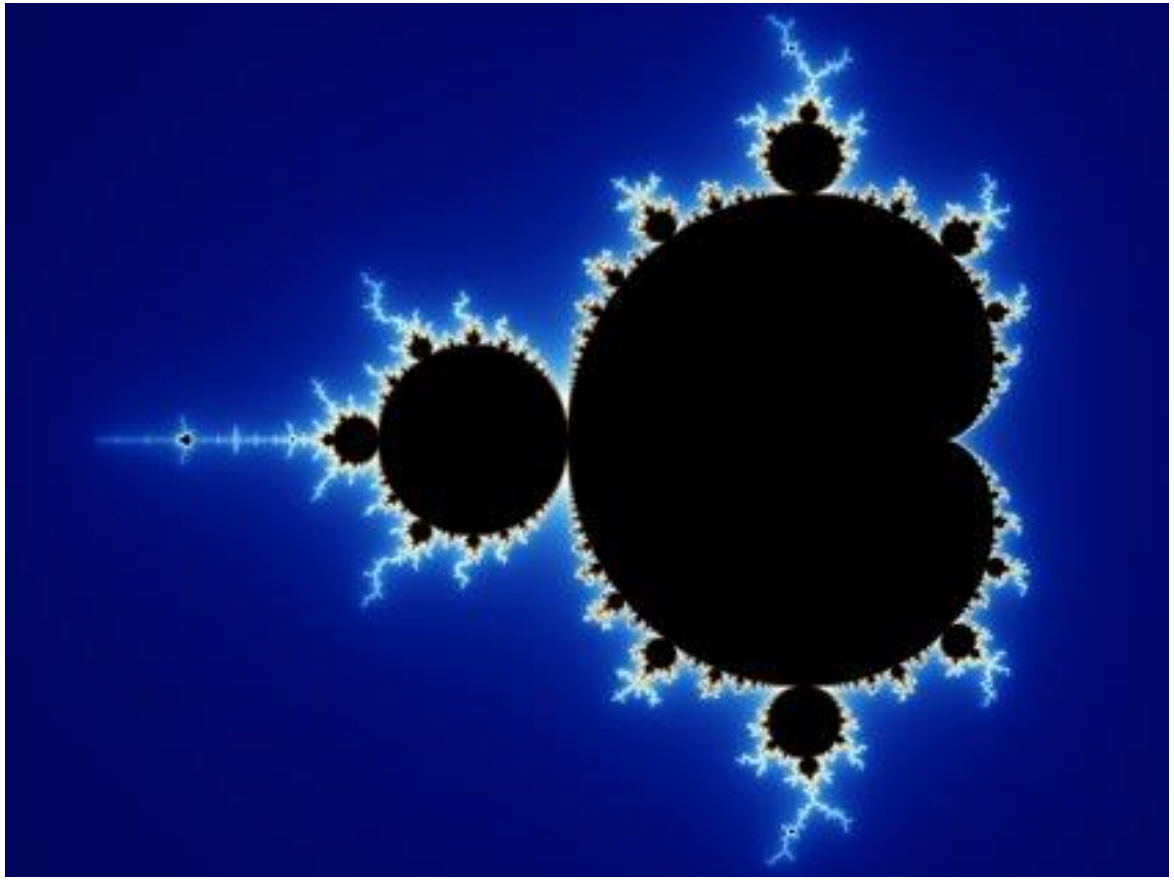


Real Python

Rekurencja

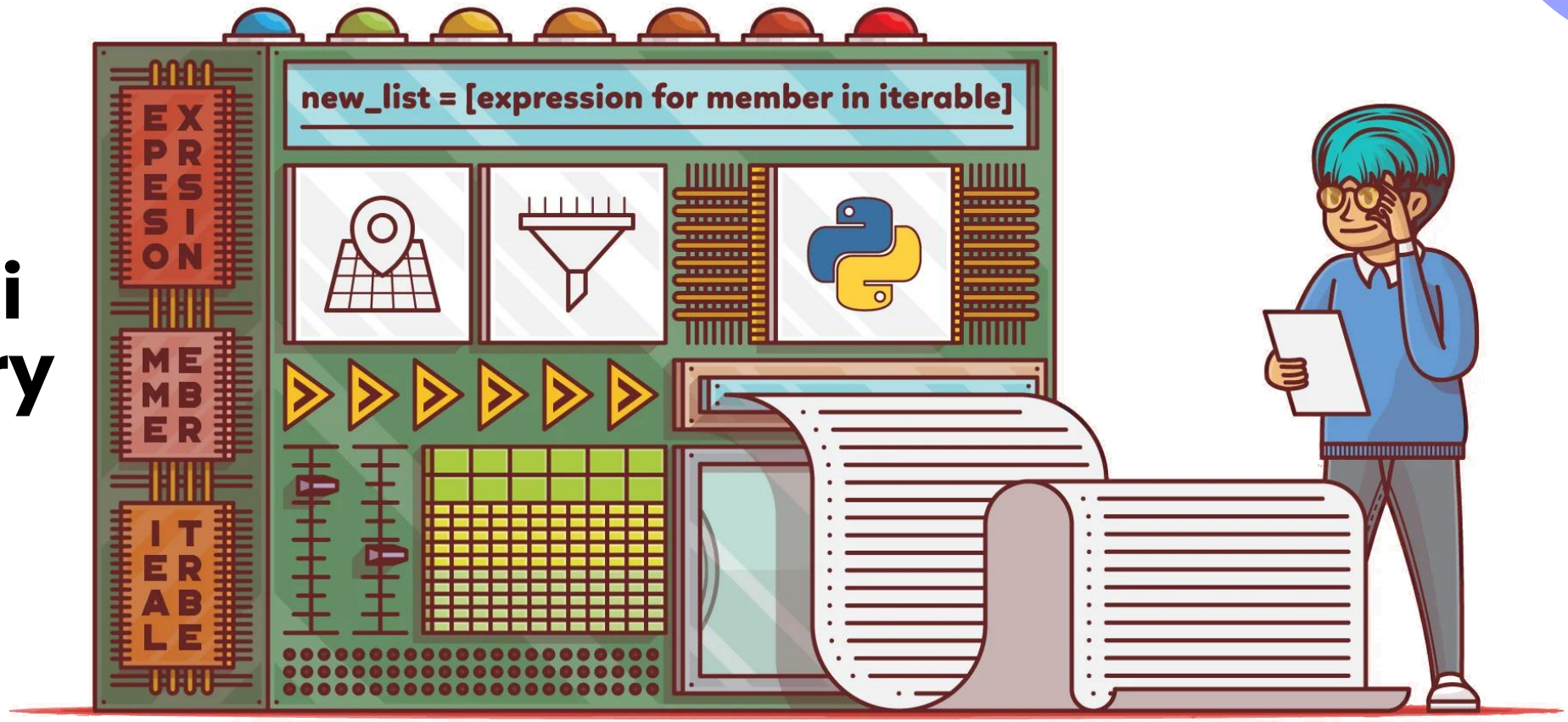
Rekurencja

Rekurencja




x == 1
def count(x):
if x == 1:
return 1
else:
return x + count(x-1)

Listy składane i generatory



Real Python

Listy składane
(list comprehension)

Tworzenie listy z wykorzystaniem zwykłej pętli

```
even_numbers = []
for item in range(10):
    if item % 2 == 0:
        even_numbers.append(item)
```

Tworzenie listy za pomocą składania (comprehension)

```
even_numbers = [item for item in range(10) if item % 2 == 0]
```

Dokumentacja

Real Python

Komentarze
(przykład)

```
# komentarz


# to jest mój komentarz
new *
def square(x):
    return x * x
```

Dostring
(przykład)

```
# docstring (notatki dokumentacyjne)


new *
def square(x):
    """Funkcja zwraca kwadrat podanej liczby"""

    return x * x


help(square)
```

Wyjątki

Obsługa wyjątków

Obsługa wyjątków

Obsługa (przechwytywanie) wyjątków oznacza reagowanie na wystąpienie błędu.
W Pythonie istnieją dwa podstawowe paradygmaty (sposoby) postępowania z błędami, spopularyzowane za pomocą akronimów:

1. **LBYL** – **L**ook **B**efore **Y**ou **L**eap
2. **EAFP** – **E**asier to **A**sk **F**orgiveness Than **P**ermission

LBYL oznacza przeciwdziałanie wystąpieniu błędu (prewencje), EAFP oznacza reagowanie na jego wystąpienie (leczenie). Odpowiedź na pytanie czy w Pythonie lepiej jest przeciwdziałać, czy leczyć nie jest tak oczywista jak to jest w medycynie. W zależności od zastosowania jeden ze sposobów przeważnie okazuje się bardziej odpowiedni od drugiego.

Wyjątki są realizacją w Python-ie tej drugiej metody postępowania - EAFP.

Obsługa wyjątków

Wróćmy do jednego z naszych pierwszych zadań.

```
from math import pi


def circle_area(r):
    """Return area od the circle for the given radius."""
    return pi * r**2


raw_r = input("Proszę, podaj mi promień koła: ")
r = float(raw_r)
area = circle_area(r)
print(f"Pole koła o promieniu {r} wynosi {round(area, 2)}")
```

Co możemy zrobić, żeby ustrzec się przed niepoprawnymi wartościami promienia podawanymi przez użytkownika ?

Obsługa wyjątków

Pierwszym pomysłem może być sprawdzanie jaką wartość przekazał użytkownik do programu (**LBYL**).

```
from math import pi


def circle_area(r):
    """Return area od the circle for the given radius."""
    return pi * r**2


raw_r = input("Proszę, podaj mi promień koła: ")

if not raw_r.isnumeric():
    print("Nie podałeś wartości liczbowej.")
else:
    r = float(raw_r)
    area = circle_area(r)
    print(f"Pole koła o promieniu {r} wynosi {round(area, 2)}")
```

Używamy instrukcji warunkowej.

W zależności od wyniku wypisujemy komunikat o niepoprawnych danych wejściowych lub wykonujemy dalej nasz program. W takim schemacie nasz warunek przyjmuje na siebie zadanie pytania o pozwolenie na wykonanie kodu (przeciwdziałamy wystąpieniu błędu).

Obsługa wyjątków

Zamiast pytać o pozwolenie możemy niczym się nie przejmując wykonać nasz program i odpowiednio zareagować jeżeli coś pójdzie nie tak (**EAFP**). Właśnie do tego celu w Python-ie wprowadzono **wyjątki**.

```
from math import pi


def circle_area(r):
    """Return area od the circle for the given radius."""
    return pi * r**2

raw_r = input("Proszę, podaj mi promień koła: ")

try:
    r = float(raw_r)
    area = circle_area(r)
    print(f"Pole koła o promieniu {r} wynosi {round(area, 2)}")

except:
    print("Nie podałeś wartości liczbowej.")
```

Używamy bloku **try-except**.

Wewnątrz instrukcji **try** umieszczamy kod, którego wywołanie może spowodować błąd (wyjątek). Wewnątrz instrukcji **except** umieszczamy kod, który ma zostać wykonany jeżeli podczas wykonywania kodu z instrukcji try wystąpi błąd (wyjątek).

W tym schemacie nie pytamy się o pozwolenie. Wykonujemy nasz kod, a jeżeli wystąpi błąd (wyjątek) odpowiednio ten błąd (wyjątek) obsługujemy (wewnątrz instrukcji except).

Pokemon i Yoda

```python
from math import pi


def circle_area(r):
    """Return area od the circle for the given radius."""
    return pi * r**2

raw_r = input("Proszę, podaj mi promień koła: ")

try:
    r = float(raw_r)
    area = circle_area(r)
    print(f"Pole koła o promieniu {r} wynosi {round(area, 2)}")

except:
    print("Nie podałeś wartości liczbowej.")
```

Pokemon exception handling - "Pokemon – gotta catch 'em all"

Yoda exception handling - "Do or do not. There is not try." (There is no partial success)

Diaper Pattern - "catches all the shit"

Typy błędów

W Pythonie mamy dwa główne typy błędów:

1. Błędy składni (Syntax Errors)
2. Wyjątki (Exceptions)

Błędy składni są to błędy wywoływane przy próbie uruchomienia programu, którego kod posiada nieprawidłową składnie. Przykładami takich błędów mogą być: nieprawidłowa głębokość wcięcia w kodzie, brak dwukropka, brak nawiasu zamykającego lub jednego z dwóch apostrofów napisu. Tego typu błędy są sygnalizowane przed właściwym uruchomieniem kodu i uniemożliwiają jego wykonanie. Kod programu ma składnie niezgodną ze składnią języka Python i program w ogóle nie może być uruchomiony za pomocą interpretera Python. Jedyny sposób naprawienia tego typu błędów to znalezienie miejsca w kodzie z nieprawidłową składnią i poprawienie go. Często pomocny jest tu traceback wyświetlany przez interpreter Pythona.

Drugi typ błędu to błędy wywoływane celowo przez twórców bibliotek w celu zasygnalizowania wystąpienia niepożądanego scenariusza. Właśnie takie błędy nazywamy **wyjątkami**. W Pythonie istnieje cała hierarchia wyjątków, której nie będziemy szczegółowo omawiać. Do najpopularniejszych (z wbudowanych) typów wyjątków należą m.in.:

- TypeError
- ValueError
- ZeroDivisionError
- IndexError
- KeyError

Python rzuca odpowiedni wyjątek w przypadku wystąpienia niepożądanego scenariusza.

Obsługa wyjątków

Pusta instrukcja except, której użyliśmy w poprzednim przykładzie jest zbyt ogólna. Znacznie lepiej jest wskazać dokładny typ wyjątku jaki chcemy obsłużyć. W naszym przykładzie Python, w wyniku błędu rzutowania na float podniesie wyjątek typu ValueError. Obsłużmy dokładnie ten typ wyjątku (a nie tak jak wcześniej - wszystkie wyjątki, które mogłyby wystąpić)

```
from math import pi


def circle_area(r):
    """Return area od the circle for the given radius."""
    return pi * r**2

raw_r = input("Proszę, podaj mi promień koła: ")

try:
    r = float(raw_r)
    area = circle_area(r)
    print(f"Pole koła o promieniu {r} wynosi {round(area, 2)}")

except ValueError:
    print("Nie podałeś wartości liczbowej.")
```

Obsługa wyjątków

W naszym kodzie, powinniśmy zadbać o właściwe obsłużenie wszystkich miejsce, w których mogą wystąpić wyjątki. Jeżeli coś poszło nie tak, użytkownik naszego programu powinien dostać czytelny komunikat co się stało.

Obsługa wyjątków

Wyjątki możemy używać również w inny sposób. Jako twórcy kodu (czyli często bibliotek) możemy je samodzielnie wywoływać w określonych scenariuszach. Wywoływanie wyjątku nazywamy rzucaniem wyjątku.

Rzucanie wyjątków

Rzucanie wyjątków

Rzucenie (podnoszenie) wyjątku oznacza celowe wywoływanie wyjątku w naszym kodzie w celu zasygnalizowania wystąpienia niepożądanego scenariusza.

W Pythonie do rzucenia wyjątku służy instrukcja raise, po której następuje nazwa rzucanego wyjątku, np.:

```
from math import pi


def circle_area(r):
    """Return area od the circle for the given radius."""

    if r == 0:
        raise ValueErorr

    return pi * r**2
```

Rzucanie wyjątków

Mówiąc o obsłudze wyjątków przyjęliśmy perspektywę użytkownika 'biblioteki'. Używamy wbudowanego kodu, który rzuca wyjątek, kiedy wystąpi niepożądany scenariusz. Naszym zadaniem jest obsługa takiego wyjątku. Drugą stroną tej samej monety jest rzucanie wyjątków.

Na to samo zagadnienie możemy patrzeć z perspektywy twórcy 'biblioteki'. Pisząc kod sami stajemy się takimi twórcami. Możliwe, że w przypadku wystąpienia niepożądanego scenariusza, my jako twórcy biblioteki sami chcielibyśmy wywołać (mówimy podnieść/rzucić) wyjątek. W takiej sytuacji odpowiedzialność za obsłużenie rzuconego przez nas wyjątku spoczywa na użytkownikach naszego kodu ('biblioteki').

Często jesteśmy twórcami oraz użytkownikami kodu jednocześnie. Na przykład piszemy funkcję (jesteśmy twórcami tej funkcji), a potem stworzoną funkcję wykorzystujemy w różnych miejscach w naszym kodzie (jesteśmy użytkownikami tej funkcji).

Rzucanie wyjątków

Po nazwie wyjątku, wewnątrz nawiasów można dodać opcjonalnych komunikat, który zostanie wyświetlony w przypadku podniesienia wyjątku (realizacji scenariusza, który wywoła wyjątek).

```
from math import pi

def circle_area(r):
    """Return area od the circle for the given radius."""

    if r == 0:
        raise ValueErorr("""Circle with the diameter 0 is not allowed""")


    return pi * r**2
```

Rzucanie wyjątków

Wyjątki rzuca się z myślą o kodzie klienckim, którego zadaniem będzie obsłużenie tego wyjątku.

```python
from math import pi


def circle_area(r):
    """Return area of the circle for the given radius."""
    if r == 0:
        raise ValueErorr("""Circle with the diameter 0 is not allowed""")
    return pi * r**2
```

Kod kliencki

```python
raw_r = input("Proszę, podaj mi promień koła: ")
try:
    r = float(raw_r)
    area = circle_area(r)
    print(f"Pole koła o promieniu {r} wynosi {round(area, 2)}")

except ValueErorr:
    print("Nie istnieje koło o promieniu 0.")
```

Rzucanie wyjątków

Wyjątki rzuca się z myślą o kodzie klienckim, którego zadaniem będzie obsłużenie tego wyjątku.

```python
from math import pi


def circle_area(r):
    """Return area of the circle for the given radius."""
    if r == 0:
        raise ValueErorr("""Circle with the diameter 0 is not allowed""")
    return pi * r**2
```

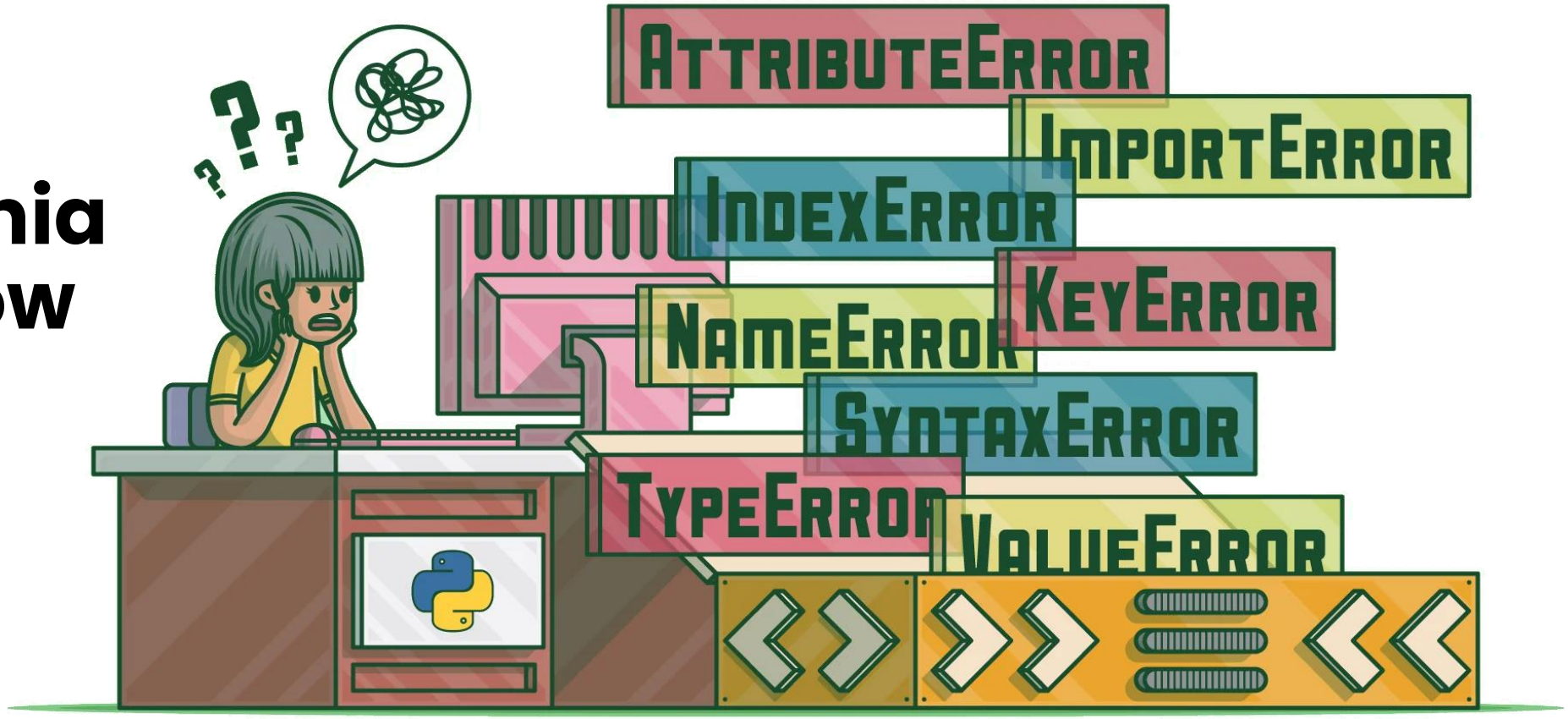
Rzucenie wyjątku

Kod kliencki

```python
raw_r = input("Proszę, podaj mi promień koła: ")

try:
    r = float(raw_r)
    area = circle_area(r)
    print(f"Pole koła o promieniu {r} wynosi {round(area, 2)}")


except ValueErorr:
    print("Nie istnieje koło o promieniu 0.")
```

Obsługa wyjątku

Hierarchia wyjątków



Real Python

Dziękujemy!

software
development
academy