Evoluce Pythonu

- Pavel Tišnovský
- kurzy.python@centrum.cz

Python

Obsah kurzu

- Nové vlastnosti jazyka
- Novinky v ekosystému Pythonu
- Vylepšení výkonnosti Pythonu
- Python a vývoj webových aplikací
- Alternativní projekty a jazyky
- Testování aplikací v Pythonu

Obsah kurzu

Nové vlastnosti jazyka

- Formátovací řetězce
- Pouze poziční parametry funkcí
- Pattern matching
- "Mroží" operátor
- Podpora pro asynchronní programování
- Skupiny výjimek
- Deklarace datových typů
- Statická typová kontrola

Obsah kurzu

Novinky v ekosystému Pythonu

- Správa projektů
- Lintery

Obsah kurzu

Vylepšení výkonnosti Pythonu

- Výkonnější CPython
- Problém související s GILem
- JIT překlad

Obsah kurzu

Python a vývoj webových aplikací

- Brython
- Transcrypt
- PyScript
- Bokeh

Obsah kurzu

Alternativní projekty a jazyky

- Coconut
- Mojo

Obsah kurzu

Testování aplikací v Pythonu

- Jednotkové testy
- Zjištění pokrytí kódu testy
- Testy chování (BDD)
- Nástroj Hypothesis
- Fuzzy testy

Nové vlastnosti jazyka

Python

Nové vlastnosti jazyka

- Formátovací řetězce
- Pouze poziční parametry funkcí
- Pattern matching
- "Mroží" operátor
- Podpora pro asynchronní programování
- Skupiny výjimek
- Deklarace datových typů
- Statická typová kontrola

Postupné rozšiřování možností Pythonu

```
Python 3.6    f-řetězce, async-IO
Python 3.7    klíčová slova async a await
Python 3.8    mroží operátor, poziční parametry
Python 3.9    generické typy
Python 3.10   pattern matching
Python 3.11   skupiny výjimek
Python 3.12   klíčové slovo type + sémantika
```

Formátovací řetězce

- Přidáno do Pythonu 3.6
- Lze využít společně s původním formátováním
- Prefix `f""`
  - proto se nazývají f-strings

Ukázka použití f-řetězců

- "Interpolace" proměnných

```
a=1
b=2
c=a+b

print(f"{a}+{b}={c}")
```

Zdrojový kód příkladu

Výrazy v f-řetězci

- V řetězci lze použít i výrazy

```
a=1
b=2

print(f"{a}+{b}={a+b}")
```

Zdrojový kód příkladu

Podmínka ve výrazu

- Ne vždy plně čitelné, ale pro jednoduché šablony ano

```
a=1
b=-1

print(f"Kladné: {'ano' if a>0 else 'ne'}")
print(f"Kladné: {'ano' if b>0 else 'ne'}")
```

Zdrojový kód příkladu

Volání funkce v f-řetězci

```
x = "Hello world!"

print(f"Délka '{x}' je {len(x)} znaků")
```

Zdrojový kód příkladu

Volání metody v f-řetězci

```
x = "hello world!"

print(f"Zpráva pro vás: '{x.capitalize()}'")
```

Zdrojový kód příkladu

Jednodušší ladění

```
name = "Guido"
surname = "Rossum"

print(f"{name=} {surname=}")
```

Zdrojový kód příkladu

Výsledek:

```
name='Guido' surname='Rossum'
```

Poziční parametry funkcí

- Přidáno do Pythonu 3.8
- Umožňují rozlišit funkce s parametry zapisovanými jen pozičně
- Ostatní parametry buď pozičně nebo je lze pojmenovat

Poziční parametry funkcí

- Běžně deklarovaná funkce

```
def foo(x, y, z):
    return x+y-z


print(foo(1, 2, 10))
print(foo(x=1, y=2, z=10))
```

Zdrojový kód příkladu

Poziční parametry funkcí

- Parametry lze pojmenovat a předat v jiném pořadí

```
def foo(x, y, z):
    return x+y-z


print(foo(1, 2, 10))
print(foo(z=1, y=2, x=10))
```

Zdrojový kód příkladu

Poziční parametry funkcí

- Všechny parametry jsou čistě poziční

```
def foo(x, y, z, /):
    return x+y-z
```

```
print(foo(1, 2, 10))
print(foo(z=1, y=2, x=10))
```

Zdrojový kód příkladu

Poziční parametry funkcí

- První parametr je čistě poziční

```
def foo(x, /, y, z):
    return x+y-z


print(foo(1, 2, 10))
print(foo(1, z=1, y=2))
```

Zdrojový kód příkladu

Poziční parametry funkcí

- Kombinace s pojmenovanými parametry

```
def foo(x=0, /, y=0, z=0):
    return x+y-z


print(foo())
print(foo(10))
print(foo(1, 2, 10))
print(foo(1, z=1, y=2))
```

Zdrojový kód příkladu

Pattern matching

- Přidáno do Pythonu 3.10
- Lepší varianta konstrukce `switch-case`

Inspirováno dalšími programovacími jazyky

- SNOBOL
- AWK
- ML (Caml, OCaml, F#)
- Rust
- Coconut (překládáno do Pythonu)

Částečně flexibilní řešení

- Ne všechny vzory je možné použít
  - například `"literal" + x + "literal"`
  - možná se jejich podpora objeví v další verzi Pythonu?

Ukázky pattern matchingu

Klasické řešení problému bez pattern matchingu

```
print("Not ready reading drive A")


def abort_retry_fail():
    response = input("Abort, Retry, Fail? ")

    if response == "a":
        return "Abort"
    elif response == "r":
        return "Retry"
    elif response == "f":
        return "Fail"
    else:
        return "Wrong response"


print(abort_retry_fail())
```

Zdrojový kód příkladu

Použití mapy (slovníku)

```
print("Not ready reading drive A")


def abort_retry_fail():
    response = input("Abort, Retry, Fail? ")

    commands = {
            "a": "Abort",
            "r": "Retry",
            "f": "Fail"
            }

    return commands.get(response, "Wrong response")


print(abort_retry_fail())
```

Zdrojový kód příkladu

Řídicí struktura match-case

```
print("Not ready reading drive A")


def abort_retry_fail():
    response = input("Abort, Retry, Fail? ")

    match response:
        case "a":
            return "Abort"
        case "r":
            return "Retry"
        case "f":
            return "Fail"
        case _:
            return "Wrong response"


print(abort_retry_fail())
```

Zdrojový kód příkladu

Množiny pro větší množství vstupů

```
print("Not ready reading drive A")


def abort_retry_fail():
    response = input("Abort, Retry, Fail? ")

    if response in {"a", "A"}:
        return "Abort"
    elif response in {"r", "R"}:
        return "Retry"
    elif response in {"f", "F"}:
        return "Fail"
    else:
        return "Wrong response"


print(abort_retry_fail())
```

Zdrojový kód příkladu

Spojka `or` ve vzoru

```
print("Not ready reading drive A")


def abort_retry_fail():
    response = input("Abort, Retry, Fail? ")

    match response:
        case "a" | "A":
            return "Abort"
        case "r" | "R":
            return "Retry"
        case "f" | "F":
            return "Fail"
        case _:
            return "Wrong response"


print(abort_retry_fail())
```

Zdrojový kód příkladu

Zachycení hodnoty ve vzoru

```
print("Not ready reading drive A")


def abort_retry_fail():
    response = input("Abort, Retry, Fail? ")

    match response:
        case "a" | "A":
            return "Abort"
        case "r" | "R":
            return "Retry"
        case "f" | "F":
            return "Fail"
        case _ as x:
            return f"Wrong response {x}"


print(abort_retry_fail())
```

Zdrojový kód příkladu

Zachycení hodnoty ve vzoru

```
print("Not ready reading drive A")


def abort_retry_fail():

    match input("Abort, Retry, Fail? "):
        case "a" | "A":
            return "Abort"
        case "r" | "R":
            return "Retry"
        case "f" | "F":
            return "Fail"
        case _ as x:
            return f"Wrong response {x}"


print(abort_retry_fail())
```

Zdrojový kód příkladu

Generátor Fibonaccího posloupnosti

```
def fib(value):
    match value:
        case 0:
            return 0
        case 1:
            return 1
        case n if n>1:
            return fib(n-1) + fib(n-2)
        case _ as wrong:
            raise ValueError("Wrong input", wrong)


for n in range(0, 11):
    print(n, fib(n))

fib(-1)
```

Zdrojový kód příkladu

Výpočet faktoriálu - základní varianta

```
def factorial(n):
    match n:
        case 0:
            return 1
        case 1:
            return 1
        case x:
            return x * factorial(x-1)


for i in range(0, 10):
    print(i, factorial(i))
```

Zdrojový kód příkladu

Podmínka ve větvi

```
def factorial(n):
    match n:
        case 0:
            return 1
        case 1:
            return 1
```

```
        case x if x>1:
            return x * factorial(x-1)
        case _:
            raise TypeError("expecting integer >= 0")


for i in range(-1, 10):
    try:
        print(i, factorial(i))
    except Exception as e:
        print(e)
```

Zdrojový kód příkladu

Test typu

```
def factorial(n):
    match n:
        case 0:
            return 1
        case 1:
            return 1
        case x if isinstance(x, int) and x>1:
            return x * factorial(x-1)
        case _:
            raise TypeError("expecting integer >= 0")


for i in range(-1, 10):
    try:
        print(i, factorial(i))
    except Exception as e:
        print(e)

try:
    print(factorial(3.14))
except Exception as e:
    print(e)

try:
    print(factorial("hello"))
except Exception as e:
    print(e)
```

Zdrojový kód příkladu

Větev "or"

```
def factorial(n):
    match n:
        case 0 | 1:
            return 1
        case x if isinstance(x, int) and x>1:
            return x * factorial(x-1)
        case _:
            raise TypeError("expecting integer >= 0")


for i in range(-1, 10):
    try:
        print(i, factorial(i))
    except Exception as e:
        print(e)

try:
    print(factorial(3.14))
except Exception as e:
    print(e)

try:
    print(factorial("hello"))
except Exception as e:
    print(e)
```

Zdrojový kód příkladu

Vzorek s n-ticí

```
def test_number(value):
    match value:
        case (0, 0):
            print("Zero")
        case (real, 0):
            print(f"Real number {real}")
        case (0, imag):
            print(f"Imaginary number {imag}")
        case (real, imag):
            print(f"Complex number {real}+i{imag}")
        case _:
            raise ValueError("Not a complex number")


test_number((0,0))
```

```
test_number((1,0))
test_number((0,1))
test_number((1,1))
```

Zdrojový kód příkladu

Vzorek s n-ticí a s podmínkou

```
def test_number(value):
    match value:
        case (0, 0):
            print("Zero")
        case (real, 0) if real>0:
            print(f"Positive real number {real}")
        case (real, 0):
            print(f"Negative real number {real}")
        case (0, imag) if imag<0:
            print(f"Negative imaginary number {imag}")
        case (0, imag):
            print(f"Negative imaginary number {imag}")
        case (real, imag):
            print(f"Complex number {real}+i{imag}")
        case _:
            raise ValueError("Not a complex number")


test_number((0,0))
test_number((1,0))
test_number((-1,0))
test_number((0,1))
test_number((0,-1))
test_number((1,1))
```

Zdrojový kód příkladu

Příkazy složené z většího množství slov

```
def perform_command():
    response = input("> ")

    match response:
        case "quit":
            return "Quit"
        case "list employees":
            return "List employees"
```

```
        case "list departments":
            return "List departments"
        case "list rooms":
            return "List rooms"
        case _:
            return "Wrong command"


print(perform_command())
```

Zdrojový kód příkladu

Vzorek a seznamy

```
def perform_command():
    response = input("> ")

    match response.split():
        case ["quit"]:
            return "Quit"
        case ["list", "employees"]:
            return "List employees"
        case ["list", "departments"]:
            return "List departments"
        case ["list", "rooms"]:
            return "List rooms"
        case _:
            return "Wrong command"


print(perform_command())
```

Zdrojový kód příkladu

Zachycení hodnoty

```
def perform_command():
    response = input("> ")

    match response.split():
        case ["quit"]:
            return "Quit"
        case ["list", "employees"]:
            return "List employees"
        case ["list", "departments"]:
```

```
            return "List departments"
        case ["list", "rooms"]:
            return "List rooms"
        case ["info", subject]:
            return f"Info about subject '{subject}'"
        case _:
            return "Wrong command"


print(perform_command())
```

Zdrojový kód příkladu

Vnořená konstrukce `match-case`

```
def perform_command():
    response = input("> ")

    match response.split():
        case ["quit"]:
            return "Quit"
        case ["list", obj]:
            match obj:
                case "employees":
                    return "List employees"
                case "departments":
                    return "List departments"
                case "rooms":
                    return "List rooms"
                case _:
                    return "Invalid object type: employees, departments, or rooms ex
        case ["info", subject]:
            return f"Info about subject '{subject}'"
        case _:
            return "Wrong command"


print(perform_command())
```

Zdrojový kód příkladu

Vnořená konstrukce `match-case` + množiny ve vzorku

```
def perform_command():
    response = input("> ")
```

```
    match response.split():
        case ["quit"]:
            return "Quit"
        case ["list", ("employees" | "departments" | "rooms") as obj]:
            match obj:
                case "employees":
                    return "List employees"
                case "departments":
                    return "List departments"
                case "rooms":
                    return "List rooms"
        case ["info", subject]:
            return f"Info about subject '{subject}'"
        case _:
            return "Wrong command"


print(perform_command())
```

Zdrojový kód příkladu

Vzorky a OOP

```
class Complex():

    def __init__(self, real, imag):
        self.real = real
        self.imag = imag

    def __str__(self):
        return f"Complex number {self.real}+i{self.imag} represented as object"


def test_number(value):
    match value:
        case (0, 0):
            print("Zero")
        case (real, 0) if real>0:
            print(f"Positive real number {real}")
        case (real, 0):
            print(f"Negative real number {real}")
        case (0, imag) if imag<0:
            print(f"Negative imaginary number {imag}")
        case (0, imag):
            print(f"Negative imaginary number {imag}")
        case (real, imag):
            print(f"Complex number {real}+i{imag}")
```

```
        case Complex():
            print(value)
        case _:
            raise ValueError("Not a complex number")


test_number((0,0))
test_number((1,0))
test_number((-1,0))
test_number((0,1))
test_number((0,-1))
test_number((1,1))

test_number(Complex(0,0))
test_number(Complex(1,0))
test_number(Complex(-1,0))
test_number(Complex(0,1))
test_number(Complex(0,-1))
test_number(Complex(1,1))
```

Zdrojový kód příkladu

Vzorky a OOP

```
from fractions import Fraction


class Complex():

    def __init__(self, real, imag):
        self.real = real
        self.imag = imag

    def __str__(self):
        return f"Complex number {self.real}+i{self.imag} represented as object"


def test_number(value):
    match value:
        case (0, 0):
            print("Zero")
        case (real, 0) if real>0:
            print(f"Positive real number {real}")
        case (real, 0):
            print(f"Negative real number {real}")
        case (0, imag) if imag<0:
            print(f"Negative imaginary number {imag}")
        case (0, imag):
```

```
            print(f"Negative imaginary number {imag}")
        case (real, imag):
            print(f"Complex number {real}+i{imag}")
        case Complex(real=0, imag=0):
            print(f"Zero complex represented as object")
        case Complex():
            print(value)
        case Fraction():
            print(f"Fraction {value}")
        case _:
            raise ValueError("Not a complex number")


test_number((0,0))
test_number((1,0))
test_number((-1,0))
test_number((0,1))
test_number((0,-1))
test_number((1,1))

test_number(Complex(0,0))
test_number(Complex(1,0))
test_number(Complex(-1,0))
test_number(Complex(0,1))
test_number(Complex(0,-1))
test_number(Complex(1,1))

test_number(Fraction(0,1))
test_number(Fraction(1,1))
test_number(Fraction(1,2))
test_number(Fraction(1,3))
```

Zdrojový kód příkladu

Mroží operátor

- Přidáno do Pythonu 3.8
- PEP 572 - Assignment Expressions
- Možnost přiřazení v rámci `výrazu`
  - původní přiřazení lze jen v rámci `příkazu`
- Takzvané `pojmenované výrazy`

Proměnná definovaná v podmínce

```
limit = 8

password = "Hello world"

if (length := len(password)) < limit:
    print(f"Password should be longer than {length} chars")
```

Zdrojový kód příkladu

Dtto, ale opačný výsledek

```
limit = 8

password = "Hello"

if (length := len(password)) < limit:
    print(f"Password should be longer than {length} chars")
```

Zdrojový kód příkladu

Problém: opakované výpočty

```
values = (1, 2, 3, 4, 5)

result = {
        "count": len(values),
        "sum": sum(values),
        "mean": sum(values) / len(values)
        }

print(result)
```

Zdrojový kód příkladu

Předpočet hodnot

```
values = (1, 2, 3, 4, 5)

count = len(values)
summ = sum(values)
```

```
result = {
        "count": count,
        "sum": summ,
        "mean": summ/count
        }

print(result)
```

Zdrojový kód příkladu

Úprava založená na walrus operátoru

```
values = (1, 2, 3, 4, 5)

result = {
        "count": (count := len(values)),
        "sum": (summ := sum(values)),
        "mean": summ/count
        }

print(result)
```

Zdrojový kód příkladu

Podpora pro asynchronní programování

- Postupně přidáno v Pythonu 3.6 a 3.7
- Nová klíčová slova `async` a `await`

Souběžnost a paralelismus

- Nejedná se o tytéž vlastnosti
- Souběžnost
  - více úloh běžících na menším množství CPU
  - i na jednom CPU
  - překrývání
- Paralelismus
  - n úloh na n CPU

Souběžnost a paralelismus v Pythonu

- Více procesů
  - `multiprocessing`
- Více vláken
  - `threading`
- Korutiny
  - `asyncio`

`async` a `await`

- Nejenom v Pythonu
  - populární i v dalších jazycích
- Typicky pro I/O operace
- Funkce označené `async`
- Čekání na dokončení pomocí `await`

`async` a `await`

- Nekorektní použití `await`

```
import asyncio
import time


async def task():
    print("task started")
    await asyncio.sleep(5)
    print("task finished")


def main():
    task1 = asyncio.create_task(task())
    print("task created")

    await task1

    print("done")


main()
```

Zdrojový kód příkladu

`async` a `await`

- Korektní použití `await`

```
import asyncio


async def task():
    print("task started")
    await asyncio.sleep(5)
    print("task finished")


async def main():
    task1 = asyncio.create_task(task())
    print("task created")

    await task1

    print("done")


asyncio.run(main())
```

Zdrojový kód příkladu

Dvě souběžné úlohy

```
import asyncio


async def task(name):
    print(f"{name} task started")
    await asyncio.sleep(5)
    print(f"{name} task finished")


async def main():
    task1 = asyncio.create_task(task("first"))
    print("first task created")

    task2 = asyncio.create_task(task("second"))
    print("second task created")
```

```
    await task1
    await task2

    print("done")


asyncio.run(main())
```

Zdrojový kód příkladu

Tři souběžné úlohy, čtení výsledné hodnoty

```
import asyncio


async def task(name):
    print(f"{name} task started")
    await asyncio.sleep(5)
    print(f"{name} task finished")
    return name[::-1]


async def main():
    task1 = asyncio.create_task(task("first"))
    print("first task created")

    task2 = asyncio.create_task(task("second"))
    print("second task created")

    task3 = asyncio.create_task(task("third"))
    print("third task created")

    print("result of task #1:", await task1)
    print("result of task #2:", await task2)
    print("result of task #3:", await task3)

    print("done")


asyncio.run(main())
```

Zdrojový kód příkladu

Komunikace přes fronty

```
import asyncio


async def task(name, queue):
    while not queue.empty():
        param = await queue.get()
        print(f"Task named {name} started with parameter {param}")
        await asyncio.sleep(5)
        print(f"{name} task finished")


async def main():
    queue = asyncio.Queue()

    for i in range(20):
        await queue.put(i)

    for n in range(1, 2):
        asyncio.create_task(task(f"{n}", queue))


asyncio.run(main())
```

Zdrojový kód příkladu

Čtení výsledků přes frontu

- Synchronizace

```
import asyncio


async def task(name, queue):
    while not queue.empty():
        param = await queue.get()
        print(f"Task named {name} started with parameter {param}")
        await asyncio.sleep(5)
        print(f"{name} task finished")


async def main():
    queue = asyncio.Queue()

    for i in range(20):
        await queue.put(i)

    for n in range(1, 2):
```

```
        await asyncio.gather(asyncio.create_task(task(f"{n}", queue)))


asyncio.run(main())
```

Zdrojový kód příkladu

Producent-konzument

- Běžící asynchronně

```
import asyncio


async def task(name, queue):
    while not queue.empty():
        param = await queue.get()
        print(f"Task named {name} started with parameter {param}")
        await asyncio.sleep(5)
        print(f"{name} task finished")


async def main():
    queue = asyncio.Queue()

    for i in range(20):
        await queue.put(i)

    await asyncio.gather(
        asyncio.create_task(task(1, queue)),
        asyncio.create_task(task(2, queue)),
        asyncio.create_task(task(3, queue)),
        asyncio.create_task(task(4, queue)),
    )


asyncio.run(main())
```

Zdrojový kód příkladu

Prioritní fronta

```
import queue
import random
```

```
q = queue.PriorityQueue(40)

for item in range(30):
    print("Size", q.qsize())
    print("Empty?", q.empty())
    print("Full?", q.full())

    value = random.randint(1, 20)
    print(value)
    q.put("prvek # {:2d}".format(value))


while not q.empty():
    print("Read item:", q.get())
```

Zdrojový kód příkladu

aiohttp

```
import asyncio
import aiohttp
import time


async def download(name, queue):
    async with aiohttp.ClientSession() as session:
        while not queue.empty():
            url = await queue.get()
            print(f"Task named {name} getting URL: {url}")
            async with session.get(url) as response:
                t = await response.text()
                print(f"Task named {name} downloaded {len(t)} characters")
            print(f"Task named {name} finished")


async def main():
    queue = asyncio.Queue()

    for url in (
        "http://www.root.cz",
        "http://duckduckgo.com",
        "http://seznam.com",
        "https://www.root.cz/programovaci-jazyky/",
        "https://www.root.cz/clanky/soubezne-a-paralelne-bezici-ulohy-naprogramovane
        "https://github.com/"
    ):
        await queue.put(url)
```

```
    await asyncio.gather(
            asyncio.create_task(download(1, queue)),
            asyncio.create_task(download(2, queue)))


asyncio.run(main())
```

Zdrojový kód příkladu

aiohttp

```
import aiohttp
import time


async def download(name, queue, results):
    async with aiohttp.ClientSession() as session:
        while not queue.empty():
            url = await queue.get()
            t1 = time.time()
            print(f"Task named {name} getting URL: {url}")
            async with session.get(url) as response:
                t = await response.text()
                t2 = time.time()
                print(f"Task named {name} downloaded {len(t)} characters in {t2-t1}
                await results.put(t2-t1)
            print(f"Task named {name} finished")


async def main():
    queue = asyncio.Queue()
    results = asyncio.Queue()

    t1 = time.time()

    for url in (
        "http://www.root.cz",
        "http://duckduckgo.com",
        "http://seznam.com",
        "https://www.root.cz/programovaci-jazyky/",
        "https://www.root.cz/clanky/soubezne-a-paralelne-bezici-ulohy-naprogramovane
        "https://www.root.cz/clanky/pywebio-interaktivni-webove-dialogy-a-formulare-
        "https://streamlit.io/",
        "https://pglet.io/",
        "https://www.root.cz/serialy/graficke-uzivatelske-rozhrani-v-pythonu/",
        "https://github.com/"
    ):
        await queue.put(url)
```

```
    await asyncio.gather(
            asyncio.create_task(download(1, queue, results)),
            asyncio.create_task(download(2, queue, results)),
            asyncio.create_task(download(3, queue, results)))

    process_time = 0
    while not results.empty():
        process_time += await results.get()

    print(f"Process time: {process_time} seconds")

    t2 = time.time()
    print(f"Total time:   {t2-t1} seconds")

asyncio.run(main())
```

Zdrojový kód příkladu

Skupiny výjimek

- Přidáno do Pythonu 3.11
- PEP 654 – Exception Groups and except

Vyhození skupiny výjimek

```
eg = ExceptionGroup(
    "one", [TypeError(1), ValueError(3), OSError(4)])

import traceback
traceback.print_exception(eg)
```

Zdrojový kód příkladu

Vyhození skupiny výjimek

```
eg = ExceptionGroup(
    "one",
    [
        TypeError(1),
        ExceptionGroup(
            "two",
```

```
            [TypeError(2), ValueError(3)]
        ),
        ExceptionGroup(
            "three",
             [OSError(4)]
        )
    ]
)

import traceback
traceback.print_exception(eg)
```

Zdrojový kód příkladu

Deklarace datových typů

- Přidáváno postupně
- PEP 484 - Type Hints a další

Nejpopulárnější jazyky současnosti

Dynamicky typované	Staticky typované
Python	C
JavaScript	C++
Ruby	Go
Perl	Rust
Matlab	Java
PHP	Scala

Přednosti dynamicky typovaných jazyků

- Rychlý cyklus vývoje
  - edit-(compile)-run
- Velmi snadné pro začátečníky
- Ideální pro skriptování
  - CLI
  - skripty na webových stránkách

Zápory dynamicky typovaných jazyků

- Zaručení korektnosti rozsáhlých projektů
- Většinou se vyžaduje větší množství jednotkových testů
  - code coverage není dobrou metrikou!
- Informace o typech se někdy zapisují do komentářů
- IDE nemusí vždy nabízet správné funkce/metody/opravy

To nejlepší z obou světů?

- Volitelné typy

```
Jazyk          Technologie pro statické typy
-------------------------------------------
JavaScript     TypeScript, Flow
Python         Mypy, Pyright, Pyre
Ruby           Sorbet
```

Volitelné typy a Python

- Python je dynamicky typovaný
  - a nejsou plány to změnit!
- Typy jsou čistě volitelné
  - přidáno do Pythonu 3.5
  - nazvané "type hints"
  - (aby to vývojáře nestrašilo)
- Statické typové kontroly
  - mypy, pyright, pyre

Statická typová kontrola a Mypy

Mypy logo

```
def add(a, b):
    return a+b
```

Zdrojový kód příkladu

- Typ `Any` je přidán automaticky

Typové anotace

- specifikují se za dvojtečkou

```
def add(a:int, b:int) -> int:
    return a+b
```

Zdrojový kód příkladu

`bool` nebo `int` ?

- Viz specifikace Pythonu!

```
def add(a:int, b:int) -> int:
    return a+b

print(add(1, 2))
print(add(1, True))
print(add(1, False))
```

Zdrojový kód příkladu

```
def add(a:bool, b:bool) -> bool:
    return a and b

print(add(1, 2))
print(add(1, True))
print(add(1, False))
print(add(True, False))
```

Zdrojový kód příkladu

Výpis typových anotací

- `any`

```
def add(a, b):
    return a+b


print(add.__annotations__)
```

Zdrojový kód příkladu

- explicitní typy

```
def add(a:int, b:int) -> int:
    return a+b


print(add.__annotations__)
```

Zdrojový kód příkladu

Výpis typových anotací

- složitější typy

```
from typing import List, Set

def add(a:List[Set[int]], b:List[Set[int]]) -> List[Set[int]]:
    return a+b


print(add.__annotations__)
```

Zdrojový kód příkladu

Typované n-tice

- nekorektní varianta

```
from typing import Tuple
```

```
p: Tuple[int] = (1, 2, 3)
```

Zdrojový kód příkladu

- korektní varianta

```
from typing import Tuple

p: Tuple[int, int, int] = (1, 2, 3)
```

Zdrojový kód příkladu

Rozdílné typy prvků

- nekorektní varianta

```
from typing import Tuple

p: Tuple[int, float, bool, str] = (1, 3.14, True, "Hello")
```

Zdrojový kód příkladu

- korektní varianta

```
from typing import Tuple

p: Tuple[int, float, bool, str] = (2.0, 3.14, 1, "Hello")
```

Zdrojový kód příkladu

Typované seznamy

- nekorektní varianta

```
l: list[int] = []
```

Zdrojový kód příkladu

- import

```
from typing import List

l: List[int] = []
```

Zdrojový kód příkladu

Test typu prvků

- v pořádku

```
from typing import List

l: List[int] = [1, 2, 3]
```

Zdrojový kód příkladu

- nekorektní

```
from typing import List

l: List[int] = [1, 2, None]
```

Zdrojový kód příkladu

Znovu problém bool-int

- v pořádku

```
from typing import List

l: List[int] = [1, True, False]
```

Zdrojový kód příkladu

- nekorektní

```
from typing import List

l: List[bool] = [True, False, 42]
```

Zdrojový kód příkladu

Typované slovníky

- Slovníky v Pythonu

```
d = {}

d["foo"] = 1
d["bar"] = 3
d["baz"] = 10

print(d)
```

Zdrojový kód příkladu

- Libovolné klíče a hodnoty

```
d = {}

d["foo"] = 1
d["bar"] = 3.14
d[10] = 10
d[42] = "answer"

print(d)
```

Zdrojový kód příkladu

Specifikace typu slovníku

- použití typu `any`

```
from typing import Dict, Any

d:Dict[Any, Any] = {}

d["foo"] = 1
d["bar"] = 3.14
d[10] = 10
d[42] = "answer"

print(d)
```

Zdrojový kód příkladu

Specifikace typu slovníku

- explicitní specifikace

```
from typing import Dict

d:Dict[str, float] = {}

d["foo"] = 1
d["bar"] = 3.14
d[10] = 10
d[42] = "answer"

print(d)
```

Zdrojový kód příkladu

Typ `union`

- Pro hodnoty

```
from typing import Dict, Union

d:Dict[str, Union[int, float, str]] = {}

d["foo"] = 1
d["bar"] = 3.14
d[10] = 10
d[42] = "answer"

print(d)
```

Zdrojový kód příkladu

Typ `union`

- Pro klíče

```
from typing import Dict, Union
```

```
d:Dict[Union[int, str], Union[int, float, str]] = {}

d["foo"] = 1
d["bar"] = 3.14
d[10] = 10
d[42] = "answer"

print(d)
```

Zdrojový kód příkladu

Typ `optional`

- Bez `optional`

```
from typing import Dict

d:Dict[str, float] = {}

d["foo"] = 1
d["bar"] = 3.14
d["baz"] = None

print(d)
```

Zdrojový kód příkladu

Typ `optional`

- S `optional`

```
from typing import Dict, Optional

d:Dict[str, Optional[float]] = {}

d["foo"] = 1
d["bar"] = 3.14
d["baz"] = None

print(d)
```

Zdrojový kód příkladu

Typy a funkce vyššího řádu

- typ `callable`

```
#!/usr/bin/env python3
# vim: set fileencoding=utf-8

#
#  (C) Copyright 2023  Pavel Tisnovsky
#
#  All rights reserved. This program and the accompanying materials
#  are made available under the terms of the Eclipse Public License v1.0
#  which accompanies this distribution, and is available at
#  http://www.eclipse.org/legal/epl-v10.html
#
#  Contributors:
#      Pavel Tisnovsky
#

from typing import Callable


def printIsPositive(condition:Callable[[float], bool]) -> None:
    if condition(5):
        print("Positive")
    else:
        print("Negative")


def positiveFloat(x:float) -> bool:
    return x > 0.0


def positiveInt(x:int) -> bool:
    return x > 0


printIsPositive(positiveFloat)
printIsPositive(positiveInt)
```

Zdrojový kód příkladu

Problém s variancí

- Týká se podtypů a nadřazených typů
  - v OOP běžné
- Čtyři možné typy variance

- kovariance
- kontravariance
- invariance
- bivariance

Příklad variancí

- `Jablko` je podtypem typu `Ovoce` ve všech dalších případech

Příklad variancí

- Covariance
  - `List[Apple]` je podtypem `List[Fruit]`
- Contravariance
  - `List[Fruit]` je podtypem `List[Apple]`
- Invariance
  - `List[Fruit]` nemá žádný vztah k `List[Apple]`
- Bivariance
  - `List[Apple]` je podtypem `List[Fruit]`
  - a současně (!!!):
  - `List[Fruit]` je podtypem `List[Apple]`

Proč se o varianci vůbec starat?

- Úzce souvisí s typovým systémem
- A s tím, jaké kontroly lze provést staticky

```
class Fruit {
}

class Orange extends Fruit {
    public String toString() {
        return "Orange";
    }
}
```

```
class Apple extends Fruit {
    public String toString() {
        return "Apple";
    }
}

public class Variance1 {
    public static void mix(Fruit[] punnet) {
        punnet[0] = new Orange();
        punnet[1] = new Apple();
    }

    public static void main(String[] args) {
        Fruit[] punnet = new Fruit[2];
        mix(punnet);

        for (Fruit Fruit:punnet) {
            System.out.println(Fruit);
        }
    }
}
```

Statická kontrola typů ok, pád v runtime!

```
class Fruit {
}

class Orange extends Fruit {
    public String toString() {
        return "Orange";
    }
}

class Apple extends Fruit {
    public String toString() {
        return "Apple";
    }
}

public class Variance2 {
    public static void mix(Fruit[] punnet) {
        punnet[0] = new Orange();
        punnet[1] = new Apple();
    }

    public static void main(String[] args) {
        Fruit[] punnet = new Orange[2];
        mix(punnet);
```

```
        for (Fruit Fruit:punnet) {
            System.out.println(Fruit);
        }
    }
}
```

Variance v Pythonu

```
from typing import List


class Ovoce:
    pass


class Hruska(Ovoce):
    def __repr__(self):
        return "Hruska"


class Jablko(Ovoce):
    def __repr__(self):
        return "Jablko"


def tiskni(kosik : List[Ovoce]):
    for ovoce in kosik:
        print(ovoce)


kosik : List[Hruska] = []

tiskni(kosik)
```

Zdrojový kód příkladu

Použití `sequence` a nikoli seznamu

```
from typing import Sequence


class Ovoce:
    pass
```

```
class Hruska(Ovoce):
    def __repr__(self):
        return "Hruska"


class Jablko(Ovoce):
    def __repr__(self):
        return "Jablko"


def tiskni(kosik : Sequence[Ovoce]):
    for ovoce in kosik:
        print(ovoce)


kosik : Sequence[Hruska] = []

tiskni(kosik)
```

Zdrojový kód příkladu

Tisk typové anotace

```
from typing import Sequence


class Ovoce:
    pass


class Hruska(Ovoce):
    def __repr__(self):
        return "Hruska"


class Jablko(Ovoce):
    def __repr__(self):
        return "Jablko"


def tiskni(kosik : Sequence[Ovoce]):
    for ovoce in kosik:
        print(ovoce)


kosik : Sequence[Hruska] = []
```

```
tiskni(kosik)

print(tiskni.__annotations__)
```

Zdrojový kód příkladu

Novinky v ekosystému Pythonu

Python

Novinky v ekosystému Pythonu

- Správa projektů
- Lintery

Správa projektů

- pip (+ venv či virtualenv)
- Pyenv
- Poetry
- Hatch
- PDM

Lintery

- pycodestyle
- pydocstyle
- black
- ruff
- (mypy)

pip

- `requirements.txt`
  - typicky zvolené verze
  - min/max/konkrétní/latest

Problémy pipu

- jak pracovat s virtualenv
- kontrola verzí (konzistence)
- řešení tranzitivních závislostí
- striktní nebo příliš volný rozsah verzí
- nejsou k dispozici otisky balíčků
  - možný prostor pro útoky
  - nelze reprodukovat "build" na dalším počítači
  - či dokonce na stejném počítači později

Další problémy

- kam uložit metadata projektu
  - nastavení linterů
  - informace o autorovi
  - aliasy příkazů
- `setup.py` je jen částečným řešením
- `setup.cfg` `.coveragerc` `tox.ini`
  - atd. atd.

`setup.cfg`

```
[metadata]
name = ccx-data-pipeline
author = somebody
description-file = README.md
license = Apache 2.0
long_description_content_type = text/markdown
home-page = https://github.com/somebody/ccx-data-pipeline
classifier =
    Intended Audience :: Information Technology
    Intended Audience :: System Administrators
```

```
    Operating System :: POSIX :: Linux
    Programming Language :: Python
    Programming Language :: Python :: 3
    Programming Language :: Python :: 3.7

[options]
zip_safe = False
packages = find:
install_requires =
    app-common-python
    insights-core-messaging
    ccx-ocp-core
    ccx-rules-ocp
    ccx-rules-ocm
    kafka-python
    requests
    jsonschema
    python-json-logger
    prometheus_client
    python-logstash
    boto3
    watchtower
setup_requires =
    setuptools
    setuptools_scm
    wheel

[options.packages.find]
exclude =
    test*

[options.entry_points]
console_scripts =
    ccx-data-pipeline = ccx_data_pipeline.command_line:ccx_data_pipeline

[options.extras_require]
dev =
    black
    coverage
    freezegun
    pycco
    pycodestyle
    pydocstyle
    pylint
    pytest
    pytest-cov

[pycodestyle]
ignore = E402
max-line-length = 100
exclude =
    .tox,
```

```
    .git,
    __pycache__,
    build,
    dist,
    tests/,
    samples/,
    *.pyc,
    *.egg-info,
    .cache,
    .eggs,
    docs,
    .venv,
    venv,

[flake8]
max-line-length = 100
```

setup.py

```
from setuptools import setup


setup(use_scm_version={"local_scheme": "node-and-timestamp"})
```

requirements.txt

```
-i https://repository.engineering.redhat.com/nexus/repository/ccx/simple

app-common-python==0.1.8
attrs==19.3.0
boto3==1.14.27
botocore==1.17.27
CacheControl==0.12.6
ccx-ocp-core==2021.12.08
ccx-rules-ocm==0.0.1
ccx-rules-ocp==2021.12.08
certifi==2020.6.20
cffi==1.14.0
chardet==3.0.4
colorama==0.4.3
cryptography==3.0
dateparser==0.7.6
decorator==4.4.2
defusedxml==0.6.0
docutils==0.15.2
```

```
fsspec==0.7.4
idna==2.10
importlib-metadata==1.7.0
insights-core>=3.0.235
insights-core-messaging==1.2.0
Jinja2==2.11.2
jmespath==0.10.0
jsonschema==3.2.0
kafka-python==2.0.1
lockfile==0.12.2
MarkupSafe==1.1.1
msgpack==1.0.0
numpy==1.19.1
packaging==20.7
pandas==1.0.5
prometheus-api-client==0.3.1
prometheus-client==0.9.0
py==1.9.0
pycparser==2.20
pyparsing==2.4.7
pyrsistent==0.16.0
python-dateutil==2.8.1
python-json-logger==0.1.11
python-logstash==0.4.6
pytz==2020.1
PyYAML==5.3.1
redis==3.5.3
regex==2020.7.14
requests==2.24.0
retry==0.9.2
retrying==1.3.3
s3fs==0.4.2
s3transfer==0.3.3
six==1.15.0
tzlocal==2.1
urllib3==1.25.10
watchtower==0.8.0
zipp==3.1.0
sentry-sdk==0.19.5
```

`pyproject.toml`

- všechna metadata v jediném souboru
- PEP-621
- správa závislostí pro různá prostředí
- metadata pro další nástroje
    - `ruff`

    - `mypy`
    - `black`

Lock file

- obsahuje konkrétní verze závislostí
- taktéž otisky balíčků
- i pro tranzitivní závislosti
- build lze kdykoli zopakovat
    - jiný počítač
    - stejný počítač v jiném okamžiku

PDM

- správce závislostí
- správce prostředí
- používá `pyproject.toml`
- a lock file

PDM

- vytvoření nového projektu
- soubor `pyproject.toml`
- přidání nové závislosti
- tranzitivní závislosti
- závislosti pro vývojáře
- lock file
- správa prostředí

Lintery

- Black
- Pycodestyle
- Pydocstyle

- Ruff

Black

- automatické formátování zdrojového kódu
- na základě specifikovaných pravidel

Pycodestyle

- kontrola, zda zdrojový kód odovídá PEP-8
- whodné zkombinovat s dalšími podobnými nástroji
  - Ruff atd.

Pydocstyle

- kontrola dokumentačních řetězců
- moduly
- třídy
- metody
- funkce

Ruff

- nový nástroj pro kontrolu zdrojových kódů Pythonu
- napsáno v Rustu
  - velmi rychlý
- možno relativně snadno přidat do CI

Ruff

- konfigurace v souboru `pyproject.toml`

```
[tool.ruff]
#select = ["E", "F", "W", "C", "D"]
select = ["E", "F", "W", "C"]
```

```
ignore = ["D211", "D213", "E402"]

line-length = 100
```

Makefile

```
style:  code-style docs-style ## Perform all style checks

code-style: ## Check code style for all Python sources from this repository
        python3 tools/run_pycodestyle.py

ruff: ## Run Ruff linter
        ruff .

docs-style: ## Check documentation strings in all Python sources from this repositor
        pydocstyle .

doc-check: ## Run gen_scenario_list.py to generate docs file and compare it to curre
        python3 tools/gen_scenario_list.py > tmp.md
        diff tmp.md docs/scenarios_list.md
```

Kontrola na CI

- konfigurace repositáře
- TravisCI
- GitHub Actions
- atd.

TravisCI

- `.travic.yml`

```
language: python
python:
    #- "3.7"
  - "3.8"
  - "3.8-dev"  # 3.8 development branch
  - "nightly"  # nightly build
addons:
  apt:
    packages:
```

```
    - libsnappy-dev
# Pycodestyle part
# needed to work correctly with Python 3 shebang
env: SKIP_INTERPRETER=true
install:
  - pip install pycodestyle
  - pip install pytest-cov
  - pip install -r requirements.txt
script:
  - make code-style
  - pytest -v --cov=schemas/
```

GitHub Actions

- .github/workflows/*.yaml

```
name: Ruff
on: [ push, pull_request ]
jobs:
  ruff:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: chartboost/ruff-action@v1
```

GitHub Actions

- .github/workflows/*.yaml

```
name: Pytest

on:
  - push
  - pull_request

jobs:
  pytest:
    runs-on: ubuntu-20.04
    strategy:
      matrix:
        python-version:
          - "3.7"
          - "3.8"
          - "3.9"
          - "3.10"
```

```
        - "3.11"
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-python@v4
        with:
          python-version: ${{ matrix.python-version }}
      - run: pip install --upgrade setuptools
      - run: pip install --upgrade wheel
      - run: pip install pycodestyle
      - run: pip install pydocstyle
      - run: pip install pytest-cov
      - run: pip install --upgrade importlib-metadata
      - run: pip install behave
      - run: pip install semver
      - name: Style checks
        run: make style
      - name: Docstrings checks
        run: make doc-check
      - name: Unit tests
        run: make unit_tests
      - name: Unit tests coverage
        run: make coverage
```

Vylepšení výkonnosti Pythonu

Python

Vylepšení výkonnosti Pythonu

- Výkonnější CPython
- Problém související s GILem
- JIT překlad

Výkonnější CPython

Problém související s GILem

JIT překlad

- Just-in-time (JIT)
- Ahead-of-time (AOT)
- Několik projektů nabízejících JIT/AOT
- Proč?
  - viz další slajdy

Problematika výkonu aplikací psaných v Pythonu

https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/python3-gcc.html

Řešený problém

- Dynamické typování + přetížené operátory
- Základní vlastnosti Pythonu

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


z = add_two_numbers(123, 456)
print(z)
```

Zdrojový kód příkladu

Jak tento problém vyřešit?

- AOT překladač
  - Cython
- JIT překladač
  - Numba

Cython

- Nadmnožina Pythonu
  - (což už neplatí)

- Překládaný jazyk
  - jedná se o transpiler do jazyka C
  - `.pyx -> .c -> .so -> launch.py`
- Explicitní datové typy jsou nepovinné
- `nogil`
- Volání nativních funkcí

Překlad do C

```
cdef add_two_numbers(x, y):
    return x + y


z = add_two_numbers(123, 456)
print(z)
```

Zdrojový kód příkladu

Explicitní typy parametrů

```
cdef add_two_numbers(int x, int y):
    return x + y


z = add_two_numbers(123, 456)
print(z)
```

Zdrojový kód příkladu

Zákaz GILu

```
cdef int add_two_numbers(int x, int y) nogil:
    return x + y


z = add_two_numbers(123, 456)
print(z)
```

Zdrojový kód příkladu

Zavolání standardní C funkce

```
from libc.stdio cimport printf


cdef int add_two_numbers(int x, int y) nogil:
    printf("%i\n", x)
    return x + y


z = add_two_numbers(123, 456)
print(z)
```

Zdrojový kód příkladu

Numba

- JIT pro Python

Dekorátor @jit

```
from numba import jit

@jit
def funkce1():
    pass
```

Zdrojový kód příkladu

Jednodušší a rychlejší `print`

- Pouze pro čísla a řetězce
- Bez nepovinných argumentů `file` a `sep`

Vynucení JITu

```
@jit(nopython=True)
```

Zdrojový kód příkladu

Porovnání výkonnosti

- ANSI C: ANSI C (ne Python)
- Cython #1: základní varianta
- Cython #2: bez typových informací
- Cython #3: optimalizace `+ nogil`
- Numba #1: původní varianta
- Numba #2: s dekorátorem `@jit`
- Numba #3: nativní funkce `print`
- Numba #4: nativní funkce `print` + @jit(nopython=True)

ANSI C: ANSI C (ne Python)

```
#include <stdlib.h>
#include <stdio.h>

#include "palette_mandmap.h"

void calc_mandelbrot(unsigned int width, unsigned int height, unsigned int maxiter,
{
    puts("P3");
    printf("%d %d\n", width, height);
    puts("255");

    double cy = -1.5;
    int y;
    for (y=0; y<height; y++) {
        double cx = -2.0;
        int x;
        for (x=0; x<width; x++) {
            double zx = 0.0;
            double zy = 0.0;
            unsigned int i = 0;
            while (i < maxiter) {
                double zx2 = zx * zx;
                double zy2 = zy * zy;
                if (zx2 + zy2 > 4.0) {
                    break;
                }
                zy = 2.0 * zx * zy + cy;
                zx = zx2 - zy2 + cx;
```

```
                i++;
            }
            unsigned char *color = palette[i];
            unsigned char r = *color++;
            unsigned char g = *color++;
            unsigned char b = *color;
            printf("%d %d %d\n", r, g, b);
            cx += 3.0/width;
        }
        cy += 3.0/height;
    }
}

int main(int argc, char **argv)
{
    if (argc < 4) {
        puts("usage: ./mandelbrot width height maxiter");
        return 1;
    }
    int width = atoi(argv[1]);
    int height = atoi(argv[2]);
    int maxiter = atoi(argv[3]);
    calc_mandelbrot(width, height, maxiter, palette);
    return 0;
}
```

Zdrojový kód příkladu

Cython #1: základní varianta

```
import palette_mandmap
from sys import argv, exit


def calc_mandelbrot(width, height, maxiter, palette):
    print("P3")
    print("{w} {h}".format(w=width, h=height))
    print("255")

    cy = -1.5
    for y in range(0, height):
        cx = -2.0
        for x in range(0, width):
            zx = 0.0
            zy = 0.0
            i = 0
            while i < maxiter:
```

```
                zx2 = zx * zx
                zy2 = zy * zy
                if zx2 + zy2 > 4.0:
                    break
                zy = 2.0 * zx * zy + cy
                zx = zx2 - zy2 + cx
                i += 1

            r = palette[i][0]
            g = palette[i][1]
            b = palette[i][2]
            print("{r} {g} {b}".format(r=r, g=g, b=b))
            cx += 3.0/width
        cy += 3.0/height


if __name__ == "__main__":
    if len(argv) < 4:
        print("usage: python mandelbrot width height maxiter")
        exit(1)

    width = int(argv[1])
    height = int(argv[2])
    maxiter = int(argv[3])
    calc_mandelbrot(width, height, maxiter, palette_mandmap.palette)
```

Zdrojový kód příkladu

Cython #2: bez typových informací

```
import palette_mandmap
from sys import argv, exit


cdef calc_mandelbrot(int width, int height, int maxiter, palette):
    cdef double zx
    cdef double zy
    cdef double zx2
    cdef double zy2
    cdef double cx
    cdef double cy
    cdef int r
    cdef int g
    cdef int b
    cdef int i

    print("P3")
    print("{w} {h}".format(w=width, h=height))
```

```
    print("255")
    cy = -1.5

    for y in range(0, height):
        cx = -2.0
        for x in range(0, width):
            zx = 0.0
            zy = 0.0
            i = 0
            while i < maxiter:
                zx2 = zx * zx
                zy2 = zy * zy
                if zx2 + zy2 > 4.0:
                    break
                zy = 2.0 * zx * zy + cy
                zx = zx2 - zy2 + cx
                i += 1

            r = palette[i][0]
            g = palette[i][1]
            b = palette[i][2]
            print("{r} {g} {b}".format(r=r, g=g, b=b))
            cx += 3.0/width
        cy += 3.0/height


if __name__ == "__main__":
    if len(argv) < 4:
        print("usage: python mandelbrot width height maxiter")
        exit(1)

    width = int(argv[1])
    height = int(argv[2])
    maxiter = int(argv[3])
    calc_mandelbrot(width, height, maxiter, palette_mandmap.palette)
```

Zdrojový kód příkladu

Cython #3: optimalizace + `nogil`

```
import palette_mandmap
from sys import argv, exit
import cython
from cpython cimport array
from libc.stdio cimport printf


@cython.cdivision(True)
```

```
cdef void calc_mandelbrot(int width, int height, int maxiter, unsigned char *palette
    cdef double zx
    cdef double zy
    cdef double zx2
    cdef double zy2
    cdef double cx
    cdef double cy
    cdef unsigned char r
    cdef unsigned char g
    cdef unsigned char b
    cdef int i
    cdef int index

    printf("P3\n%d %d\n255\n", width, height)
    cy = -1.5

    for y in range(0, height):
        cx = -2.0
        for x in range(0, width):
            zx = 0.0
            zy = 0.0
            i = 0
            while i < maxiter:
                zx2 = zx * zx
                zy2 = zy * zy
                if zx2 + zy2 > 4.0:
                    break
                zy = 2.0 * zx * zy + cy
                zx = zx2 - zy2 + cx
                i += 1

            index = i * 3
            r = palette[index]
            g = palette[index+1]
            b = palette[index+2]
            printf("%d %d %d\n", r, g, b)
            cx += 3.0/width
        cy += 3.0/height


cdef array.array palette = array.array('B')

if __name__ == "__main__":

    if len(argv) < 4:
        print("usage: python mandelbrot width height maxiter")
        exit(1)

    for color in palette_mandmap.palette:
        for component in color:
            palette.append(component)
```

```
    width = int(argv[1])
    height = int(argv[2])
    maxiter = int(argv[3])
    calc_mandelbrot(width, height, maxiter, palette.data.as_uchars)
```

Zdrojový kód příkladu

Numba #1: původní varianta

```
import palette_mandmap
from sys import argv, exit

def calc_mandelbrot(width, height, maxiter, palette):
    print("P3")
    print("{w} {h}".format(w=width, h=height))
    print("255")

    cy = -1.5
    for y in range(0, height):
        cx = -2.0
        for x in range(0, width):
            zx = 0.0
            zy = 0.0
            i = 0
            while i < maxiter:
                zx2 = zx * zx
                zy2 = zy * zy
                if zx2 + zy2 > 4.0:
                    break
                zy = 2.0 * zx * zy + cy
                zx = zx2 - zy2 + cx
                i += 1

            r = palette[i][0]
            g = palette[i][1]
            b = palette[i][2]
            print("{r} {g} {b}".format(r=r, g=g, b=b))
            cx += 3.0/width
        cy += 3.0/height


if __name__ == "__main__":
    if len(argv) < 4:
        width = 512
        height = 512
        maxiter = 255
    else:
        width = int(argv[1])
```

```
        height = int(argv[2])
        maxiter = int(argv[3])
    calc_mandelbrot(width, height, maxiter, palette_mandmap.palette)
```

Zdrojový kód příkladu

Numba #2: s dekorátorem `@jit`

```
import palette_mandmap
from sys import argv, exit

from numba import jit


@jit
def calc_mandelbrot(width, height, maxiter, palette):
    print("P3")
    print("{w} {h}".format(w=width, h=height))
    print("255")

    cy = -1.5
    for y in range(0, height):
        cx = -2.0
        for x in range(0, width):
            zx = 0.0
            zy = 0.0
            i = 0
            while i < maxiter:
                zx2 = zx * zx
                zy2 = zy * zy
                if zx2 + zy2 > 4.0:
                    break
                zy = 2.0 * zx * zy + cy
                zx = zx2 - zy2 + cx
                i += 1

            r = palette[i][0]
            g = palette[i][1]
            b = palette[i][2]
            print("{r} {g} {b}".format(r=r, g=g, b=b))
            cx += 3.0/width
        cy += 3.0/height


if __name__ == "__main__":
    if len(argv) < 4:
        width = 512
        height = 512
```

```
        maxiter = 255
    else:
        width = int(argv[1])
        height = int(argv[2])
        maxiter = int(argv[3])
    calc_mandelbrot(width, height, maxiter, palette_mandmap.palette)
```

Zdrojový kód příkladu

Numba #3: nativní funkce `print`

```
import palette_mandmap
from sys import argv, exit

from numba import jit


@jit
def calc_mandelbrot(width, height, maxiter, palette):
    print("P3")
    print(width)
    print(height)
    print("255")

    cy = -1.5
    for y in range(0, height):
        cx = -2.0
        for x in range(0, width):
            zx = 0.0
            zy = 0.0
            i = 0
            while i < maxiter:
                zx2 = zx * zx
                zy2 = zy * zy
                if zx2 + zy2 > 4.0:
                    break
                zy = 2.0 * zx * zy + cy
                zx = zx2 - zy2 + cx
                i += 1

            r = palette[i][0]
            g = palette[i][1]
            b = palette[i][2]
            print(r)
            print(g)
            print(b)
            cx += 3.0/width
        cy += 3.0/height
```

```
if __name__ == "__main__":
    if len(argv) < 4:
        width = 512
        height = 512
        maxiter = 255
    else:
        width = int(argv[1])
        height = int(argv[2])
        maxiter = int(argv[3])
    calc_mandelbrot(width, height, maxiter, palette_mandmap.palette)
```

Zdrojový kód příkladu

Numba #4: nativní funkce `print` + @jit(nopython=True)

```
import palette_mandmap
from sys import argv, exit

from numba import jit


@jit(nopython=True)
def calc_mandelbrot(width, height, maxiter, palette):
    print("P3")
    print(width)
    print(height)
    print("255")

    cy = -1.5
    for y in range(0, height):
        cx = -2.0
        for x in range(0, width):
            zx = 0.0
            zy = 0.0
            i = 0
            while i < maxiter:
                zx2 = zx * zx
                zy2 = zy * zy
                if zx2 + zy2 > 4.0:
                    break
                zy = 2.0 * zx * zy + cy
                zx = zx2 - zy2 + cx
                i += 1

            r = palette[i][0]
            g = palette[i][1]
```

```
            b = palette[i][2]
            print(r)
            print(g)
            print(b)
            cx += 3.0/width
        cy += 3.0/height


if __name__ == "__main__":
    if len(argv) < 4:
        width = 512
        height = 512
        maxiter = 255
    else:
        width = int(argv[1])
        height = int(argv[2])
        maxiter = int(argv[3])
    calc_mandelbrot(width, height, maxiter, palette_mandmap.palette)
```

Zdrojový kód příkladu

Výsledky benchmarků 1/2

images/benchmarks-1.png

Výsledky benchmarků 2/2

images/benchmarks-2.png

Python a vývoj webových aplikací

Python

Python a vývoj webových aplikací

- Brython
- Transcrypt
- PyScript
- Bokeh

Transpilery

- "Code to code compilers"
  - transformace kódu mezi dvěma jazyky
  - použito právě pro konverzi do JavaScriptu
- AOT nebo JIT

Mnoho typů transpilerů

#	Jazyk či transpřekladač	Poznámka
1	CoffeeScript	přidání syntaktického cukru do JavaScriptu
2	ClojureScript	překlad aplikací psaných v Clojure do JavaScriptu
3	TypeScript	nadmnožina jazyka JavaScript, přidání datových typů
4	6to5	transpřeklad z ECMAScript 6 (nová varianta JavaScriptu)
5	Kaffeine	rozšíření JavaScriptu o nové vlastnosti
6	RedScript	jazyk inspirovaný Ruby
7	GorillaScript	další rozšíření JavaScriptu
8	ghcjs	transpřekladač pro fanoušky programovacího jazyka Haskel
9	Haxe	transpřekladač, mezi jehož cílové jazyka patří i Java a
10	Wisp	transpřekladač jazyka podobného Clojure, opět do JavaScr
11	ScriptSharp	transpřekladač z C# do JavaScriptu
12	Dart	transpřekladač z jazyka Dart do JavaScriptu
13	COBOL → C	transpřekladač OpenCOBOL
14	COBOL → Java	transpřekladač P3COBOL
15	lua2js	transpřekladač jazyka Lua, opět do JavaScriptu
16	Coconut	transpřekladač jazyka Coconut do Pythonu

Brython

- Transpiler Python -> JavaScript
- JIT
  - kód se překládá až při inicialiazi stránky
  - jakékoli úpravy se ihned projeví po F5

Transcrypt

- Transpiler Python -> JavaScript
- AOT

    - výsledný JS lze načíst do webové stránky
- Podpora DOM
- `print` na konzoli
    - plus většina standardních funkcí Pythonu
- Malý runtime
    - cca 20kB
- Podpora Numscryptu
    - (nedokonalá) varianta Numpy

Základní datové typy (seznamy)

```
x = [1, 2, 3, 4, 5]

x.append(99)

print(x)

for item in x:
    print(item)
```

Zdrojový kód příkladu

Základní datové typy (seznamy)

```
// Transcrypt'ed from Python, 2023-10-19 16:36:48
import {AssertionError, AttributeError, BaseException, DeprecationWarning, Exceptior
var __name__ = '__main__';
export var x = [1, 2, 3, 4, 5];
x.append (99);
print (x);
for (var item of x) {
    print (item);
}

//# sourceMappingURL=lists.map```

[Zdrojový kód příkladu](https://github.com/tisnik/most-popular-python-libs/blob/mast

---

### Základní datové typy (slovníky)
```

```python
x = {"foo": 1, "bar": 2, "baz": None}

print(x)

for key, value in enumerate(x):
    print(key, value)
```

Zdrojový kód příkladu

Základní datové typy (slovníky)

```
// Transcrypt'ed from Python, 2023-10-19 16:36:27
import {AssertionError, AttributeError, BaseException, DeprecationWarning, Exceptior
var __name__ = '__main__';
export var x = dict ({'foo': 1, 'bar': 2, 'baz': null});
print (x);
for (var [key, value] of enumerate (x)) {
    print (key, value);
}

//# sourceMappingURL=maps.map```

[Zdrojový kód příkladu](https://github.com/tisnik/most-popular-python-libs/blob/mast

---

### Funkce

```python
def add(a, b):
    return a+b
```

Zdrojový kód příkladu

Funkce

```
// Transcrypt'ed from Python, 2023-10-19 16:39:41
import {AssertionError, AttributeError, BaseException, DeprecationWarning, Exceptior
var __name__ = '__main__';
export var add = function (a, b) {
    return a + b;
};

//# sourceMappingURL=adder1.map```
```
```

```
[Zdrojový kód příkladu](https://github.com/tisnik/most-popular-python-libs/blob/mast

---

### Uzávěry

```python
def createCounter():
    counter = 0
    def next():
        nonlocal counter
        counter += 1
        return counter
    return next



#
# Spusteni testu.
#
def main():
    counter1 = createCounter()
    counter2 = createCounter()
    for i in range(1,11):
        result1 = counter1()
        result2 = counter2()
        print("Iteration #%d" % i)
        print("    Counter1: %d" % result1)
        print("    Counter2: %d" % result2)


main()
```

Zdrojový kód příkladu

Uzávěry

```
// Transcrypt'ed from Python, 2023-10-19 16:42:46
import {AssertionError, AttributeError, BaseException, DeprecationWarning, Exceptior
var __name__ = '__main__';
export var createCounter = function () {
    var counter = 0;
    var py_next = function () {
        counter++;
        return counter;
    };
    return py_next;
```
```

```
};
export var main = function () {
    var counter1 = createCounter ();
    var counter2 = createCounter ();
    for (var i = 1; i < 11; i++) {
        var result1 = counter1 ();
        var result2 = counter2 ();
        print (__mod__ ('Iteration #%d', i));
        print (__mod__ ('    Counter1: %d', result1));
        print (__mod__ ('    Counter2: %d', result2));
    }
};
main ();

//# sourceMappingURL=counter_closure.map```

[Zdrojový kód příkladu](https://github.com/tisnik/most-popular-python-libs/blob/mast

---

### Komunikace s webovou stránkou

* Skript v Pythonu

```python
from itertools import chain

class SolarSystem:
    planets = [list (chain (planet, (index + 1,))) for index, planet in enumerate ((
        ('Mercury', 'hot', 2240),
        ('Venus', 'sulphurous', 6052),
        ('Earth', 'fertile', 6378),
        ('Mars', 'reddish', 3397),
        ('Jupiter', 'stormy', 71492),
        ('Saturn', 'ringed', 60268),
        ('Uranus', 'cold', 25559),
        ('Neptune', 'very cold', 24766)
    ))]

    lines = (
        '{} is a {} planet',
        'The radius of {} is {} km',
        '{} is planet nr. {} counting from the sun'
    )

    def __init__ (self):
        self.lineIndex = 0

    def greet (self):
        self.planet = self.planets [int (Math.random () * len (self.planets))]
        document.getElementById ('greet') .innerHTML = 'Hello {}'.format (self.plane
        self.explain ()
```
```

```
    def explain (self):
        document.getElementById ('explain').innerHTML = (
            self.lines [self.lineIndex] .format (self.planet [0], self.planet [self.
        )
        self.lineIndex = (self.lineIndex + 1) % 3

solarSystem = SolarSystem ()
```

Zdrojový kód příkladu

Komunikace s webovou stránkou

- Výsledek transpřekladu

```
// Transcrypt'ed from Python, 2023-10-19 16:43:26
import {AssertionError, AttributeError, BaseException, DeprecationWarning, Exceptior
import {chain} from './itertools.js';
var __name__ = '__main__';
export var SolarSystem =  __class__ ('SolarSystem', [object], {
    __module__: __name__,
    planets: (function () {
        var __accu0__ = [];
        for (var [index, planet] of enumerate (tuple ([tuple (['Mercury', 'hot', 224
            __accu0__.append (list (chain (planet, tuple ([index + 1]))));
        }
        return __accu0__;
    }) (),
    lines: tuple (['{} is a {} planet', 'The radius of {} is {} km', '{} is planet r
    get __init__ () {return __get__ (this, function (self) {
        self.lineIndex = 0;
    });},
    get greet () {return __get__ (this, function (self) {
        self.planet = self.planets [int (Math.random () * len (self.planets))];
        document.getElementById ('greet').innerHTML = 'Hello {}'.format (self.planet
        self.explain ();
    });},
    get explain () {return __get__ (this, function (self) {
        document.getElementById ('explain').innerHTML = self.lines [self.lineIndex].
        self.lineIndex = __mod__ (self.lineIndex + 1, 3);
    });}
});
export var solarSystem = SolarSystem ();

//# sourceMappingURL=hello.map```

[Zdrojový kód příkladu](https://github.com/tisnik/most-popular-python-libs/blob/mast
```

```
---

### Kreslení na canvas

* Skript v Pythonu

```python
canvas = document.getElementById('canvas')
context = canvas.getContext('2d')

context.font = '60pt Arial'
context.fillStyle = 'darkblue'
context.strokeStyle = 'navyblue'

context.fillText('Hello Canvas', canvas.width / 2 - 210, canvas.height / 2 + 15)
context.strokeText('Hello Canvas', canvas.width / 2 - 210, canvas.height / 2 + 15)
```

Zdrojový kód příkladu

Kreslení na canvas

- Výsledek transpřekladu

```
// Transcrypt'ed from Python, 2023-10-20 14:52:48
import {AssertionError, AttributeError, BaseException, DeprecationWarning, Exceptior
var __name__ = '__main__';
export var canvas = document.getElementById ('canvas');
export var context = canvas.getContext ('2d');
context.font = '60pt Arial';
context.fillStyle = 'darkblue';
context.strokeStyle = 'navyblue';
context.fillText ('Hello Canvas', canvas.width / 2 - 210, canvas.height / 2 + 15);
context.strokeText ('Hello Canvas', canvas.width / 2 - 210, canvas.height / 2 + 15);

//# sourceMappingURL=canvas1.map```

[Zdrojový kód příkladu](https://github.com/tisnik/most-popular-python-libs/blob/mast

---

### Kreslení na canvas

* Podpůrná HTML stránka s canvasem

```python
<html>
    <head>
        <title>Canvas</title>
```

```
        <style>
            body {
              background: #dddddd;
            }

            #canvas {
              margin: 10px;
              padding: 10px;
              background: #ffffff;
              border: thin inset #aaaaaa;
            }
        </style>
    </head>
    <body>
        <canvas id='canvas' width='800' height='600'>Canvas not supported</canvas>
        <script type="module">import * as canvas from "./__target__/transcrypt-canva
        </script>
    </body>
</html>
```

Zdrojový kód příkladu

Alternativní projekty a jazyky

Python

Alterntivní projekty a jazyky

- Coconut
- Mojo

Coconut

- Jazyk překládaný do JavaScriptu
- Nové jazykové konstrukce
- Vylepšené jazykové konstrukce

Hello world

```
"Hello world!" |> print
```

Zdrojový kód příkladu

Hello world

```
(print)("Hello world!")
```

Zdrojový kód příkladu

Sekvence

```
print(map(lambda x: x*2, [1, 2, 3, 4]))
print(fmap(lambda x: x*2, [1, 2, 3, 4]))
print(map(lambda x: x*2, (1, 2, 3, 4)))
print(fmap(lambda x: x*2, (1, 2, 3, 4)))
print(map(lambda x: x*2, range(10)))
print(fmap(lambda x: x*2, range(10)))
print(reduce(lambda acc, x: acc * x, range(1, 10)))
print(list(takewhile(lambda x: x<10, range(100))))
print(list(dropwhile(lambda x: x<10, range(100))))
print(list(takewhile(lambda x: x<10, (count()))))
print(list(takewhile(lambda x: x<10, (count(0)))))
print(list(takewhile(lambda x: x<10, (count(0,2)))))
```

Zdrojový kód příkladu

Sekvence

```
print(map(lambda x: x * 2, [1, 2, 3, 4]))

print(fmap(lambda x: x * 2, [1, 2, 3, 4]))

print(map(lambda x: x * 2, (1, 2, 3, 4)))

print(fmap(lambda x: x * 2, (1, 2, 3, 4)))

print(map(lambda x: x * 2, range(10)))

print(fmap(lambda x: x * 2, range(10)))

print(reduce(lambda acc, x: acc * x, range(1, 10)))

print(list(takewhile(lambda x: x < 10, range(100))))

print(list(dropwhile(lambda x: x < 10, range(100))))

print(list(takewhile(lambda x: x < 10, (count()))))

print(list(takewhile(lambda x: x < 10, (count(0)))))

print(list(takewhile(lambda x: x < 10, (count(0, 2)))))
```

Zdrojový kód příkladu

Anonymní funkce

```
print(list(map(x -> x * 2, [1, 2, 3])))

print(fmap(x -> x * 2, [1, 2, 3]))

print(reduce( (acc,x) -> acc*x, range(1,10)))

print(list(map( (x,y,z) -> [x,y,z], [1,2,3], [4,5,6], [7,8,9])))

print(reduce( (acc, x) -> acc * x, range(1, 10)))

print(fmap(x -> x*2, range(10)))

print(list(takewhile(x -> x<10, range(100))))

print(list(dropwhile(x -> x<10, range(100))))

print(list(takewhile(x -> x<10, (count()))))
```

```
print(list(takewhile(x -> x<10, (count(0)))))

print(list(takewhile(x -> x<10, (count(0,2)))))
```

Zdrojový kód příkladu

Anonymní funkce

```
print(list(map(lambda x: x * 2, [1, 2, 3])))

print(fmap(lambda x: x * 2, [1, 2, 3]))

print(reduce(lambda acc, x: acc * x, range(1, 10)))

print(list(map(lambda x, y, z: [x, y, z], [1, 2, 3], [4, 5, 6], [7, 8, 9])))

print(reduce(lambda acc, x: acc * x, range(1, 10)))

print(fmap(lambda x: x * 2, range(10)))

print(list(takewhile(lambda x: x < 10, range(100))))

print(list(dropwhile(lambda x: x < 10, range(100))))

print(list(takewhile(lambda x: x < 10, (count()))))

print(list(takewhile(lambda x: x < 10, (count(0)))))

print(list(takewhile(lambda x: x < 10, (count(0, 2)))))
```

Zdrojový kód příkladu

Neměnitelné datové typy

```
data complex(real, imag):

    def __abs__(self) =
        (self.real**2 + self.imag**2)**1/2

    def __neg__(self) =
        (self.real, self.imag) |> map$(-) |*> complex

    def __add__(self, other) =
```

```
        complex(self.real + other.real, self.imag + other.imag)


c1=complex(1.0, 2.0)

print(-c1)

print(c1)

print(abs(c1))

c1 |> abs |> print

c2=complex(100.0, 50.0)

print(c1+c2)

print(c1+c1)
```

Zdrojový kód příkladu

Neměnitelné datové typy

```
class complex(_coconut.collections.namedtuple("complex", "real imag")):
    __slots__ = ()
    __ne__ = _coconut.object.__ne__
    def __abs__(self):
        return (self.real**2 + self.imag**2)**1 / 2

    @_coconut_tco
    def __neg__(self):
        return _coconut_tail_call((complex), *map(_coconut_minus, (self.real, self.i

    @_coconut_tco
    def __add__(self, other):
        return _coconut_tail_call(complex, self.real + other.real, self.imag + other


c1 = complex(1.0, 2.0)

print(-c1)

print(c1)

print(abs(c1))

(print)((abs)(c1))
```

```
c2 = complex(100.0, 50.0)

print(c1 + c2)

print(c1 + c1)
```

Zdrojový kód příkladu

Infixová notace

```
print("hello" `isinstance` str)

def factorial(n):
    if n <= 1:
        return 1
    else:
        return range(1, n+1) |> reduce$(*)

def choose(n, k):
    return factorial(n)/(factorial(k)*factorial(n-k))

print(factorial(10))

for k in range(5):
    print(choose(4, k))

print()

for k in range(5):
    print(4 `choose` k)

def n `nad` k:
    return factorial(n)/(factorial(k)*factorial(n-k))

print()

for k in range(5):
    print(4 `nad` k)
```

Zdrojový kód příkladu

Infixová notace

```
print((isinstance)("hello", str))

@_coconut_tco
def factorial(n):
    if n <= 1:
        return 1
    else:
        return _coconut_tail_call(reduce, _coconut.operator.mul, range(1, n + 1))

def choose(n, k):
    return factorial(n) / (factorial(k) * factorial(n - k))

print(factorial(10))

for k in range(5):
    print(choose(4, k))

print()

for k in range(5):
    print((choose)(4, k))

def nad(n, k):
    return factorial(n) / (factorial(k) * factorial(n - k))

print()

for k in range(5):
    print((nad)(4, k))
```

Zdrojový kód příkladu

Kolony

```
-42 |> abs |> print

"B" |> ord |> abs |> hex |> print

range(11) |> sum |> print

range(11) |> reversed |> sum |> print

def evens(sequence):
    return filter(lambda x: x % 2 == 0, sequence)

[1, 2, 3, 4, 5, 6, 30] |> evens |> sum |> print
```

Zdrojový kód příkladu

Kolony: po překladu

```
# Compiled Coconut: ------------------------------------------------------------

(print)((abs)(-42))

(print)((hex)((abs)((ord)("B"))))

(print)((sum)(range(11)))

(print)((sum)((reversed)(range(11))))


@_coconut_tco
def evens(sequence):
    return _coconut_tail_call(filter, lambda x: x % 2 == 0, sequence)


(print)((sum)((evens)([1, 2, 3, 4, 5, 6, 30])))
```

Zdrojový kód příkladu

Kolony: původní kód

```
# Compiled Coconut: ------------------------------------------------------------

(print)((abs)(-42))  # line 1: -42 |> abs |> print

(print)((hex)((abs)((ord)("B"))))  # line 3: "B" |> ord |> abs |> hex |> print

(print)((sum)(range(11)))  # line 5: range(11) |> sum |> print

(print)((sum)((reversed)(range(11))))  # line 7: range(11) |> reversed |> sum |> pri


@_coconut_tco  # line 9: def evens(sequence):
def evens(sequence):  # line 9: def evens(sequence):
    return _coconut_tail_call(
        filter, lambda x: x % 2 == 0, sequence
    )  # line 10:     return filter(lambda x: x % 2 == 0, sequence)
```

```
(print)(
    (sum)((evens)([1, 2, 3, 4, 5, 6, 30]))
)  # line 12: [1, 2, 3, 4, 5, 6, 30] |> evens |> sum |> print
```

Zdrojový kód příkladu

Kolony: čísla řádků

```
# Compiled Coconut: ------------------------------------------------------------

(print)((abs)(-42))  # line 1

(print)((hex)((abs)((ord)("B"))))  # line 3

(print)((sum)(range(11)))  # line 5

(print)((sum)((reversed)(range(11))))  # line 7


@_coconut_tco  # line 9
def evens(sequence):  # line 9
    return _coconut_tail_call(filter, lambda x: x % 2 == 0, sequence)  # line 10


(print)((sum)((evens)([1, 2, 3, 4, 5, 6, 30])))  # line 12
```

Zdrojový kód příkladu

Kompozice funkcí

```
"B" |> ord |> abs |> hex |> print

"B" |> hex..abs..ord |> print

range(11) |> reversed |> sum |> print

range(11) |> sum..reversed |> print

def evens(sequence):
    return filter(x -> x % 2 == 0, sequence)

[1, 2, 3, 4, 5, 6, 30] |> sum..evens |> print
```

Zdrojový kód příkladu

Kompozice funkcí

```
(print)((hex)((abs)((ord)("B"))))

(print)((_coconut_forward_compose(ord, abs, hex))("B"))

(print)((sum)((reversed)(range(11))))

(print)((_coconut_forward_compose(reversed, sum))(range(11)))

@_coconut_tco
def evens(sequence):
    return _coconut_tail_call(filter, lambda x: x % 2 == 0, sequence)

(print)((_coconut_forward_compose(evens, sum))([1, 2, 3, 4, 5, 6, 30]))
```

Zdrojový kód příkladu

Zřetězení operací

```
def generator1():
    values = ["a1", "b1", "c1", "d1"]
    for value in values:
        yield value

def generator2():
    values = ["a2", "b2", "c2", "d2"]
    for value in values:
        yield value

for v in generator1()::generator2():
    print(v)

def generator3(suffix):
    values = ["a", "b", "c", "d", "e"]
    for value in values:
        yield "{v}{s}".format(v=value, s=suffix)

for v in generator3("1")::generator3("2")::generator3("3"):
    print(v)
```

Zdrojový kód příkladu

Zřetězení operací

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# __coconut_hash__ = 0xaec2ab88

# Compiled with Coconut version 1.3.0 [Dead Parrot]

# Coconut Header: --------------------------------------------------------------

# Compiled Coconut: ------------------------------------------------------------


def generator1():
    values = ["a1", "b1", "c1", "d1"]
    for value in values:
        yield value


def generator2():
    values = ["a2", "b2", "c2", "d2"]
    for value in values:
        yield value


for v in _coconut.itertools.chain.from_iterable(
    (f() for f in (lambda: generator1(), lambda: generator2()))
):
    print(v)


def generator3(suffix):
    values = ["a", "b", "c", "d", "e"]
    for value in values:
        yield "{v}{s}".format(v=value, s=suffix)


for v in _coconut.itertools.chain.from_iterable(
    (
        f()
        for f in (
            lambda: generator3("1"),
            lambda: generator3("2"),
            lambda: generator3("3"),
        )
    )
):
```

```
    print(v)
```

Zdrojový kód příkladu

Operátor ??

```
import os

v1 = None
v2 = "Some"

print(v1 ?? v2)

v3 = "Some"
v4 = "Something else"

print(v3 ?? v4)

v5 = None
v6 = None

print(v5 ?? v6)

print(os.getenv('EDITOR') ?? "notepad")
print(os.getenv('XXEDITOR') ?? "notepad")
```

Zdrojový kód příkladu

Operátor ??

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# __coconut_hash__ = 0x109f876f

# Compiled with Coconut version 1.3.0 [Dead Parrot]

# Coconut Header: --------------------------------------------------------------

# Compiled Coconut: ------------------------------------------------------------

import os

v1 = None
v2 = "Some"
```

```
print(v2 if v1 is None else v1)

v3 = "Some"
v4 = "Something else"

print(v4 if v3 is None else v3)

v5 = None
v6 = None

print(v6 if v5 is None else v5)

print(
    (
        lambda _coconut_none_coalesce_item: "notepad"
        if _coconut_none_coalesce_item is None
        else _coconut_none_coalesce_item
    )(os.getenv("EDITOR"))
)
print(
    (
        lambda _coconut_none_coalesce_item: "notepad"
        if _coconut_none_coalesce_item is None
        else _coconut_none_coalesce_item
    )(os.getenv("XXEDITOR"))
)
```

Zdrojový kód příkladu

Operátor ??=

```
slovnik = {
    "prvni": 1,
    "druhy": 2,
    "treti": 3,
    "posledni": None
}

print(slovnik)

slovnik["prvni"] ??= 1000

print(slovnik)

slovnik["posledni"] ??= 1000

print(slovnik)
```

```
slovnik["neexistujici"] ??= 10

print(slovnik)
```

Zdrojový kód příkladu

Operátor ??=

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# __coconut_hash__ = 0xbe97918d

# Compiled with Coconut version 1.3.0 [Dead Parrot]

# Coconut Header: --------------------------------------------------------------


# Compiled Coconut: ------------------------------------------------------------

slovnik = {"prvni": 1, "druhy": 2, "treti": 3, "posledni": None}

print(slovnik)

slovnik["prvni"] = 1000 if slovnik["prvni"] is None else slovnik["prvni"]

print(slovnik)

slovnik["posledni"] = 1000 if slovnik["posledni"] is None else slovnik["posledni"]

print(slovnik)

slovnik["neexistujici"] = (
    10 if slovnik["neexistujici"] is None else slovnik["neexistujici"]
)

print(slovnik)
```

Zdrojový kód příkladu

Elvisův operátor

```
game1 = {
    "player" : {
```

```
        "name": "Kvido",
        "nick": "kvido"
    },
    "results": {
        "score": {
            "last": 1000,
            "top": 2000
        },
        "lives": 0
    }
}

def print_last_score_variant_A(game):
    score = game.get("results").get("score").get("last")
    print("Score: {s}".format(s=score))

print("\nVariant A")
print_last_score_variant_A(game1)

game2 = {
    "player" : {
        "name": "Kvido",
        "nick": "kvido"
    }
}

def print_last_score_variant_B(game):
    score = game.get("results", {}).get("score", {}).get("last")
    print("Score: {s}".format(s=score))

print("\nVariant B")
print_last_score_variant_B(game1)
print_last_score_variant_B(game2)

def print_last_score_variant_C(game):
    score = game.get("results")?.get("score")?.get("last")
    print("Score: {s}".format(s=score))

print("\nVariant C")
print_last_score_variant_C(game1)
print_last_score_variant_C(game2)

class Player:
    def __init__(self, name, nick):
        self.name = name
        self.nick = nick

class Score:
    def __init__(self,last, top):
        self.last = last
        self.top =top
```

```
class Game:
    def __init__(self,player, score, lives):
        self.player = player
        self.score = score
        self.lives = lives

def print_last_score_variant_D(game):
    score = game?.score?.last
    print("Score: {s}".format(s=score))

game1_obj = Game(Player("Kvido", "kvido"),
                 Score(1000, 2000),
                 0)

game2_obj = Game(Player("Kvido", "kvido"),
                 None, 0)

print("\nVariant D")
print_last_score_variant_D(game1_obj)
print_last_score_variant_D(game2_obj)
```

Zdrojový kód příkladu

Elvisův operátor

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# __coconut_hash__ = 0xd796c34e

# Compiled with Coconut version 1.3.0 [Dead Parrot]

# Coconut Header: --------------------------------------------------------------


# Compiled Coconut: ------------------------------------------------------------

game1 = {
    "player": {"name": "Kvido", "nick": "kvido"},
    "results": {"score": {"last": 1000, "top": 2000}, "lives": 0},
}


def print_last_score_variant_A(game):
    score = game.get("results").get("score").get("last")
    print("Score: {s}".format(s=score))
```

```
print("\nVariant A")
print_last_score_variant_A(game1)

game2 = {"player": {"name": "Kvido", "nick": "kvido"}}


def print_last_score_variant_B(game):
    score = game.get("results", {}).get("score", {}).get("last")
    print("Score: {s}".format(s=score))


print("\nVariant B")
print_last_score_variant_B(game1)
print_last_score_variant_B(game2)


def print_last_score_variant_C(game):
    score = (
        lambda x: None
        if x is None
        else (lambda x: None if x is None else x.get("last"))(x.get("score"))
    )(game.get("results"))
    print("Score: {s}".format(s=score))


print("\nVariant C")
print_last_score_variant_C(game1)
print_last_score_variant_C(game2)


class Player(_coconut.object):
    def __init__(self, name, nick):
        self.name = name
        self.nick = nick


class Score(_coconut.object):
    def __init__(self, last, top):
        self.last = last
        self.top = top


class Game(_coconut.object):
    def __init__(self, player, score, lives):
        self.player = player
        self.score = score
        self.lives = lives


def print_last_score_variant_D(game):
    score = (
        lambda x: None
```

```
        if x is None
        else (lambda x: None if x is None else x.last)(x.score)
    )(game)
    print("Score: {s}".format(s=score))


game1_obj = Game(Player("Kvido", "kvido"), Score(1000, 2000), 0)

game2_obj = Game(Player("Kvido", "kvido"), None, 0)

print("\nVariant D")
print_last_score_variant_D(game1_obj)
print_last_score_variant_D(game2_obj)
```

Zdrojový kód příkladu

Podpora Unicode

```
result = 60·7÷10
print(result)

if result ≥ 40 and result ≤ 50:
    print("very close")

print(1 « 10)
print(1 ⊕ 255)

-42 ↦ abs ↦ print

"B" ↦ ord ↦ abs ↦ hex ↦ print

range(11) ↦ sum ↦ print

range(11) ↦ reversed ↦ sum ↦ print

"B" ↦ hex ∘ abs ∘ ord ↦ print

range(11) ↦ sum ∘ reversed ↦ print
```

Zdrojový kód příkladu

Podpora Unicode

```
#!/usr/bin/env python
```

```
# -*- coding: utf-8 -*-
# __coconut_hash__ = 0x3d5464d

# Compiled with Coconut version 1.3.0 [Dead Parrot]

# Coconut Header: --------------------------------------------------------------


# Compiled Coconut: ------------------------------------------------------------

result = 60 * 7 / 10
print(result)

if result >= 40 and result <= 50:
    print("very close")

print(1 << 10)
print(1 ^ 255)

(print)((abs)(-42))

(print)((hex)((abs)((ord)("B"))))

(print)((sum)(range(11)))

(print)((sum)((reversed)(range(11))))

(print)((_coconut_forward_compose(ord, abs, hex))("B"))

(print)((_coconut_forward_compose(reversed, sum))(range(11)))
```

Zdrojový kód příkladu

Pattern matching

```
def factorial_variant_A(n):
    case n:
        match 0:
            return 1
        match 1:
            return 1
        match x:
            return x * factorial_variant_A(x-1)
    else:
        raise TypeError("expecting integer >= 0")

for n in range(11):
    print("{n}!={f}".format(n=n, f=factorial_variant_A(n)))
```

```
def factorial_variant_B(n):
    case n:
        match 0:
            return 1
        match 1:
            return 1
        match x if x > 1:
            return x * factorial_variant_B(x-1)
    else:
        raise TypeError("expecting integer >= 0")

for n in range(11):
    print("{n}!={f}".format(n=n, f=factorial_variant_B(n)))

def factorial_variant_C(n):
    case n:
        match 0:
            return 1
        match 1:
            return 1
        match x is int if x > 1:
            return x * factorial_variant_C(x-1)
    else:
        raise TypeError("expecting integer >= 0")

for n in range(11):
    print("{n}!={f}".format(n=n, f=factorial_variant_C(n)))
```

Zdrojový kód příkladu

Pattern matching

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# __coconut_hash__ = 0x2ef22af1

# Compiled with Coconut version 1.3.0 [Dead Parrot]

# Coconut Header: --------------------------------------------------------------

from __future__ import print_function, absolute_import, unicode_literals, division
import sys as _coconut_sys

if _coconut_sys.version_info < (3,):
    (
        py_chr,
```

```
    py_filter,
    py_hex,
    py_input,
    py_int,
    py_map,
    py_object,
    py_oct,
    py_open,
    py_print,
    py_range,
    py_str,
    py_zip,
    py_filter,
    py_reversed,
    py_enumerate,
) = (
    chr,
    filter,
    hex,
    input,
    int,
    map,
    object,
    oct,
    open,
    print,
    range,
    str,
    zip,
    filter,
    reversed,
    enumerate,
)
py_raw_input, py_xrange = raw_input, xrange
(
    _coconut_NotImplemented,
    _coconut_raw_input,
    _coconut_xrange,
    _coconut_int,
    _coconut_long,
    _coconut_print,
    _coconut_str,
    _coconut_unicode,
    _coconut_repr,
) = (NotImplemented, raw_input, xrange, int, long, print, str, unicode, repr)
from future_builtins import *

chr, str = unichr, unicode
from io import open

class object(object):
    __slots__ = ()
```

```
    def __ne__(self, other):
        eq = self == other
        if eq is _coconut_NotImplemented:
            return eq
        return not eq

class int(_coconut_int):
    __slots__ = ()
    if hasattr(_coconut_int, "__doc__"):
        __doc__ = _coconut_int.__doc__

    class __metaclass__(type):
        def __instancecheck__(cls, inst):
            return _coconut.isinstance(inst, (_coconut_int, _coconut_long))

        def __subclasscheck__(cls, subcls):
            return _coconut.issubclass(subcls, (_coconut_int, _coconut_long))

class range(object):
    __slots__ = ("_xrange",)
    if hasattr(_coconut_xrange, "__doc__"):
        __doc__ = _coconut_xrange.__doc__

    def __init__(self, *args):
        self._xrange = _coconut_xrange(*args)

    def __iter__(self):
        return _coconut.iter(self._xrange)

    def __reversed__(self):
        return _coconut.reversed(self._xrange)

    def __len__(self):
        return _coconut.len(self._xrange)

    def __contains__(self, elem):
        return elem in self._xrange

    def __getitem__(self, index):
        if _coconut.isinstance(index, _coconut.slice):
            args = _coconut.slice(*self._args)
            start, stop, step, ind_step = (
                (args.start if args.start is not None else 0),
                args.stop,
                (args.step if args.step is not None else 1),
                (index.step if index.step is not None else 1),
            )
            return self.__class__(
                (start if ind_step >= 0 else stop - step)
                if index.start is None
                else start + step * index.start
```

```
                if index.start >= 0
                else stop + step * index.start,
                (stop if ind_step >= 0 else start - step)
                if index.stop is None
                else start + step * index.stop
                if index.stop >= 0
                else stop + step * index.stop,
                step if index.step is None else step * index.step,
            )
        else:
            return self._xrange[index]

    def count(self, elem):
        """Count the number of times elem appears in the range."""
        return _coconut_int(elem in self._xrange)

    def index(self, elem):
        """Find the index of elem in the range."""
        if elem not in self._xrange:
            raise _coconut.ValueError(_coconut.repr(elem) + " is not in range")
        start, _, step = self._xrange.__reduce_ex__(2)[1]
        return (elem - start) // step

    def __repr__(self):
        return _coconut.repr(self._xrange)[1:]

    @property
    def _args(self):
        return self._xrange.__reduce__()[1]

    def __reduce_ex__(self, protocol):
        return (self.__class__, self._xrange.__reduce_ex__(protocol)[1])

    def __reduce__(self):
        return self.__reduce_ex__(_coconut.pickle.DEFAULT_PROTOCOL)

    def __hash__(self):
        return _coconut.hash(self._args)

    def __copy__(self):
        return self.__class__(*self._args)

    def __eq__(self, other):
        return (
            _coconut.isinstance(other, self.__class__) and self._args == other._
        )

from collections import Sequence as _coconut_Sequence

_coconut_Sequence.register(range)
from functools import wraps as _coconut_wraps
```

```
@_coconut_wraps(_coconut_print)
def print(*args, **kwargs):
    file = kwargs.get("file", _coconut_sys.stdout)
    flush = kwargs.get("flush", False)
    if "flush" in kwargs:
        del kwargs["flush"]
    if _coconut.hasattr(file, "encoding") and file.encoding is not None:
        _coconut_print(
            *(_coconut_unicode(x).encode(file.encoding) for x in args), **kwargs
        )
    else:
        _coconut_print(*(_coconut_unicode(x).encode() for x in args), **kwargs)
    if flush:
        file.flush()

@_coconut_wraps(_coconut_raw_input)
def input(*args, **kwargs):
    if (
        _coconut.hasattr(_coconut_sys.stdout, "encoding")
        and _coconut_sys.stdout.encoding is not None
    ):
        return _coconut_raw_input(*args, **kwargs).decode(
            _coconut_sys.stdout.encoding
        )
    return _coconut_raw_input(*args, **kwargs).decode()

@_coconut_wraps(_coconut_repr)
def repr(obj):
    if isinstance(obj, _coconut_unicode):
        return _coconut_unicode(_coconut_repr(obj)[1:])
    if isinstance(obj, _coconut_str):
        return "b" + _coconut_unicode(_coconut_repr(obj))
    return _coconut_unicode(_coconut_repr(obj))

ascii = repr

def raw_input(*args):
    """Coconut uses Python 3 "input" instead of Python 2 "raw_input"."""
    raise _coconut.NameError(
        'Coconut uses Python 3 "input" instead of Python 2 "raw_input"'
    )

def xrange(*args):
    """Coconut uses Python 3 "range" instead of Python 2 "xrange"."""
    raise _coconut.NameError(
        'Coconut uses Python 3 "range" instead of Python 2 "xrange"'
    )

if _coconut_sys.version_info < (2, 7):
    import functools as _coconut_functools, copy_reg as _coconut_copy_reg

    def _coconut_new_partial(func, args, keywords):
```

```
            return _coconut_functools.partial(
                func,
                *(args if args is not None else ()),
                **(keywords if keywords is not None else {})
            )

        _coconut_copy_reg.constructor(_coconut_new_partial)

        def _coconut_reduce_partial(self):
            return (_coconut_new_partial, (self.func, self.args, self.keywords))

        _coconut_copy_reg.pickle(_coconut_functools.partial, _coconut_reduce_partial
else:
    (
        py_chr,
        py_filter,
        py_hex,
        py_input,
        py_int,
        py_map,
        py_object,
        py_oct,
        py_open,
        py_print,
        py_range,
        py_str,
        py_zip,
        py_filter,
        py_reversed,
        py_enumerate,
    ) = (
        chr,
        filter,
        hex,
        input,
        int,
        map,
        object,
        oct,
        open,
        print,
        range,
        str,
        zip,
        filter,
        reversed,
        enumerate,
    )


class _coconut(object):
    import collections, copy, functools, imp, itertools, operator, types, weakref
```

```
if _coconut_sys.version_info < (3,):
    import cPickle as pickle
else:
    import pickle
if _coconut_sys.version_info >= (2, 7):
    OrderedDict = collections.OrderedDict
else:
    OrderedDict = dict
if _coconut_sys.version_info < (3, 3):
    abc = collections
else:
    import collections.abc as abc
(
    Exception,
    IndexError,
    KeyError,
    NameError,
    TypeError,
    ValueError,
    StopIteration,
    classmethod,
    dict,
    enumerate,
    filter,
    frozenset,
    getattr,
    hasattr,
    hash,
    id,
    int,
    isinstance,
    issubclass,
    iter,
    len,
    list,
    map,
    min,
    max,
    next,
    object,
    property,
    range,
    reversed,
    set,
    slice,
    str,
    sum,
    super,
    tuple,
    zip,
    repr,
```

```
        bytearray,
    ) = (
        Exception,
        IndexError,
        KeyError,
        NameError,
        TypeError,
        ValueError,
        StopIteration,
        classmethod,
        dict,
        enumerate,
        filter,
        frozenset,
        getattr,
        hasattr,
        hash,
        id,
        int,
        isinstance,
        issubclass,
        iter,
        len,
        list,
        map,
        min,
        max,
        next,
        object,
        property,
        range,
        reversed,
        set,
        slice,
        str,
        sum,
        super,
        tuple,
        zip,
        staticmethod(repr),
        bytearray,
    )


def _coconut_NamedTuple(name, fields):
    return _coconut.collections.namedtuple(name, [x for x, t in fields])


class MatchError(Exception):
    """Pattern-matching error. Has attributes .pattern and .value."""

    __slots__ = ("pattern", "value")
```

```
class _coconut_tail_call(object):
    __slots__ = ("func", "args", "kwargs")

    def __init__(self, func, *args, **kwargs):
        self.func, self.args, self.kwargs = func, args, kwargs


_coconut_tco_func_dict = {}


def _coconut_tco(func):
    @_coconut.functools.wraps(func)
    def tail_call_optimized_func(*args, **kwargs):
        call_func = func
        while True:
            wkref = _coconut_tco_func_dict.get(_coconut.id(call_func))
            if wkref is not None and wkref() is call_func:
                call_func = call_func._coconut_tco_func
            result = call_func(
                *args, **kwargs
            )  # pass --no-tco to clean up your traceback
            if not isinstance(result, _coconut_tail_call):
                return result
            call_func, args, kwargs = result.func, result.args, result.kwargs

    tail_call_optimized_func._coconut_tco_func = func
    _coconut_tco_func_dict[
        _coconut.id(tail_call_optimized_func)
    ] = _coconut.weakref.ref(tail_call_optimized_func)
    return tail_call_optimized_func


def _coconut_igetitem(iterable, index):
    if isinstance(
        iterable,
        (
            _coconut_reversed,
            _coconut_map,
            _coconut.filter,
            _coconut.zip,
            _coconut_enumerate,
            _coconut_count,
            _coconut.abc.Sequence,
        ),
    ):
        return iterable[index]
    if not _coconut.isinstance(index, _coconut.slice):
        if index < 0:
            return _coconut.collections.deque(iterable, maxlen=-index)[0]
        return _coconut.next(_coconut.itertools.islice(iterable, index, index + 1))
```

```
    if (
        index.start is not None
        and index.start < 0
        and (index.stop is None or index.stop < 0)
        and index.step is None
    ):
        queue = _coconut.collections.deque(iterable, maxlen=-index.start)
        if index.stop is not None:
            queue = _coconut.tuple(queue)[: index.stop - index.start]
        return queue
    if (
        (index.start is not None and index.start < 0)
        or (index.stop is not None and index.stop < 0)
        or (index.step is not None and index.step < 0)
    ):
        return _coconut.tuple(iterable)[index]
    return _coconut.itertools.islice(iterable, index.start, index.stop, index.step)


class _coconut_base_compose(object):
    __slots__ = ("func", "funcstars")

    def __init__(self, func, *funcstars):
        self.func = func
        self.funcstars = []
        for f, star in funcstars:
            if isinstance(f, _coconut_base_compose):
                self.funcstars.append((f.func, star))
                self.funcstars += f.funcstars
            else:
                self.funcstars.append((f, star))

    def __call__(self, *args, **kwargs):
        arg = self.func(*args, **kwargs)
        for f, star in self.funcstars:
            arg = f(*arg) if star else f(arg)
        return arg

    def __repr__(self):
        return (
            _coconut.repr(self.func)
            + " "
            + " ".join(
                ("..*> " if star else "..> ") + _coconut.repr(f)
                for f, star in self.funcstars
            )
        )

    def __reduce__(self):
        return (self.__class__, (self.func,) + _coconut.tuple(self.funcstars))
```

```
def _coconut_forward_compose(func, *funcs):
    return _coconut_base_compose(func, *((f, False) for f in funcs))


def _coconut_back_compose(*funcs):
    return _coconut_forward_compose(*_coconut.reversed(funcs))


def _coconut_forward_star_compose(func, *funcs):
    return _coconut_base_compose(func, *((f, True) for f in funcs))


def _coconut_back_star_compose(*funcs):
    return _coconut_forward_star_compose(*_coconut.reversed(funcs))


def _coconut_pipe(x, f):
    return f(x)


def _coconut_star_pipe(xs, f):
    return f(*xs)


def _coconut_back_pipe(f, x):
    return f(x)


def _coconut_back_star_pipe(f, xs):
    return f(*xs)


def _coconut_bool_and(a, b):
    return a and b


def _coconut_bool_or(a, b):
    return a or b


def _coconut_none_coalesce(a, b):
    return a if a is not None else b


def _coconut_minus(a, *rest):
    if not rest:
        return -a
    for b in rest:
        a = a - b
    return a
```

```
@_coconut.functools.wraps(_coconut.itertools.tee)
def tee(iterable, n=2):
    if n >= 0 and _coconut.isinstance(iterable, (_coconut.tuple, _coconut.frozenset)
        return (iterable,) * n
    if n > 0 and (
        _coconut.hasattr(iterable, "__copy__")
        or _coconut.isinstance(iterable, _coconut.abc.Sequence)
    ):
        return (iterable,) + _coconut.tuple(
            _coconut.copy.copy(iterable) for _ in _coconut.range(n - 1)
        )
    return _coconut.itertools.tee(iterable, n)


class reiterable(object):
    """Allows an iterator to be iterated over multiple times."""

    __slots__ = ("iter",)

    def __init__(self, iterable):
        self.iter = iterable

    def __iter__(self):
        self.iter, out = _coconut_tee(self.iter)
        return _coconut.iter(out)

    def __getitem__(self, index):
        return _coconut_igetitem(_coconut.iter(self), index)

    def __reversed__(self):
        return _coconut_reversed(_coconut.iter(self))

    def __len__(self):
        return _coconut.len(self.iter)

    def __repr__(self):
        return "reiterable(" + _coconut.repr(self.iter) + ")"

    def __reduce__(self):
        return (self.__class__, (self.iter,))

    def __copy__(self):
        return self.__class__(_coconut.copy.copy(self.iter))

    def __fmap__(self, func):
        return _coconut_map(func, self)


class scan(object):
    """Reduce func over iterable, yielding intermediate results."""

    __slots__ = ("func", "iter")
```

```
    def __init__(self, func, iterable):
        self.func, self.iter = func, iterable

    def __iter__(self):
        acc = empty_acc = _coconut.object()
        for item in self.iter:
            if acc is empty_acc:
                acc = item
            else:
                acc = self.func(acc, item)
            yield acc

    def __len__(self):
        return _coconut.len(self.iter)

    def __repr__(self):
        return "scan(" + _coconut.repr(self.iter) + ")"

    def __reduce__(self):
        return (self.__class__, (self.func, self.iter))

    def __copy__(self):
        return self.__class__(self.func, _coconut.copy.copy(self.iter))

    def __fmap__(self, func):
        return _coconut_map(func, self)


class reversed(object):
    __slots__ = ("_iter",)
    if hasattr(_coconut.map, "__doc__"):
        __doc__ = _coconut.reversed.__doc__

    def __new__(cls, iterable):
        if _coconut.isinstance(iterable, _coconut.range):
            return iterable[::-1]
        if not _coconut.hasattr(iterable, "__reversed__") or _coconut.isinstance(
            iterable, (_coconut.list, _coconut.tuple)
        ):
            return _coconut.object.__new__(cls)
        return _coconut.reversed(iterable)

    def __init__(self, iterable):
        self._iter = iterable

    def __iter__(self):
        return _coconut.iter(_coconut.reversed(self._iter))

    def __getitem__(self, index):
        if _coconut.isinstance(index, _coconut.slice):
            return _coconut_igetitem(
```

```
                self._iter,
                _coconut.slice(
                    -(index.start + 1) if index.start is not None else None,
                    -(index.stop + 1) if index.stop else None,
                    -(index.step if index.step is not None else 1),
                ),
            )
        return _coconut_igetitem(self._iter, -(index + 1))

    def __reversed__(self):
        return self._iter

    def __len__(self):
        return _coconut.len(self._iter)

    def __repr__(self):
        return "reversed(" + _coconut.repr(self._iter) + ")"

    def __hash__(self):
        return -_coconut.hash(self._iter)

    def __reduce__(self):
        return (self.__class__, (self._iter,))

    def __copy__(self):
        return self.__class__(_coconut.copy.copy(self._iter))

    def __eq__(self, other):
        return isinstance(other, self.__class__) and self._iter == other._iter

    def __contains__(self, elem):
        return elem in self._iter

    def count(self, elem):
        """Count the number of times elem appears in the reversed iterator."""
        return self._iter.count(elem)

    def index(self, elem):
        """Find the index of elem in the reversed iterator."""
        return _coconut.len(self._iter) - self._iter.index(elem) - 1

    def __fmap__(self, func):
        return self.__class__(_coconut_map(func, self._iter))


class map(_coconut.map):
    __slots__ = ("_func", "_iters")
    if hasattr(_coconut.map, "__doc__"):
        __doc__ = _coconut.map.__doc__

    def __new__(cls, function, *iterables):
        new_map = _coconut.map.__new__(cls, function, *iterables)
```

```
        new_map._func, new_map._iters = function, iterables
        return new_map

    def __getitem__(self, index):
        if _coconut.isinstance(index, _coconut.slice):
            return self.__class__(
                self._func, *(_coconut_igetitem(i, index) for i in self._iters)
            )
        return self._func(*(_coconut_igetitem(i, index) for i in self._iters))

    def __reversed__(self):
        return self.__class__(self._func, *(_coconut_reversed(i) for i in self._iter

    def __len__(self):
        return _coconut.min(_coconut.len(i) for i in self._iters)

    def __repr__(self):
        return (
            "map("
            + _coconut.repr(self._func)
            + ", "
            + ", ".join((_coconut.repr(i) for i in self._iters))
            + ")"
        )

    def __reduce__(self):
        return (self.__class__, (self._func,) + self._iters)

    def __reduce_ex__(self, _):
        return self.__reduce__()

    def __copy__(self):
        return self.__class__(
            self._func, *_coconut.map(_coconut.copy.copy, self._iters)
        )

    def __fmap__(self, func):
        return self.__class__(_coconut_forward_compose(self._func, func), *self._ite


class parallel_map(map):
    """Multi-process implementation of map using concurrent.futures.
    Requires arguments to be pickleable."""

    __slots__ = ()

    def __iter__(self):
        from concurrent.futures import ProcessPoolExecutor

        with ProcessPoolExecutor() as executor:
            return _coconut.iter(_coconut.tuple(executor.map(self._func, *self._iter
```

```
    def __repr__(self):
        return "parallel_" + _coconut_map.__repr__(self)


class concurrent_map(map):
    """Multi-thread implementation of map using concurrent.futures."""

    __slots__ = ()

    def __iter__(self):
        from concurrent.futures import ThreadPoolExecutor
        from multiprocessing import (
            cpu_count,
        )  # cpu_count() * 5 is the default Python 3.5 thread count

        with ThreadPoolExecutor(cpu_count() * 5) as executor:
            return _coconut.iter(_coconut.tuple(executor.map(self._func, *self._iter

    def __repr__(self):
        return "concurrent_" + _coconut_map.__repr__(self)


class filter(_coconut.filter):
    __slots__ = ("_func", "_iter")
    if hasattr(_coconut.filter, "__doc__"):
        __doc__ = _coconut.filter.__doc__

    def __new__(cls, function, iterable):
        new_filter = _coconut.filter.__new__(cls, function, iterable)
        new_filter._func, new_filter._iter = function, iterable
        return new_filter

    def __reversed__(self):
        return self.__class__(self._func, _coconut_reversed(self._iter))

    def __repr__(self):
        return (
            "filter("
            + _coconut.repr(self._func)
            + ", "
            + _coconut.repr(self._iter)
            + ")"
        )

    def __reduce__(self):
        return (self.__class__, (self._func, self._iter))

    def __reduce_ex__(self, _):
        return self.__reduce__()

    def __copy__(self):
        return self.__class__(self._func, _coconut.copy.copy(self._iter))
```

```
    def __fmap__(self, func):
        return _coconut_map(func, self)


class zip(_coconut.zip):
    __slots__ = ("_iters",)
    if hasattr(_coconut.zip, "__doc__"):
        __doc__ = _coconut.zip.__doc__

    def __new__(cls, *iterables):
        new_zip = _coconut.zip.__new__(cls, *iterables)
        new_zip._iters = iterables
        return new_zip

    def __getitem__(self, index):
        if _coconut.isinstance(index, _coconut.slice):
            return self.__class__(*(_coconut_igetitem(i, index) for i in self._iters
        return _coconut.tuple(_coconut_igetitem(i, index) for i in self._iters)

    def __reversed__(self):
        return self.__class__(*(_coconut_reversed(i) for i in self._iters))

    def __len__(self):
        return _coconut.min(_coconut.len(i) for i in self._iters)

    def __repr__(self):
        return "zip(" + ", ".join((_coconut.repr(i) for i in self._iters)) + ")"

    def __reduce__(self):
        return (self.__class__, self._iters)

    def __reduce_ex__(self, _):
        return self.__reduce__()

    def __copy__(self):
        return self.__class__(*_coconut.map(_coconut.copy.copy, self._iters))

    def __fmap__(self, func):
        return _coconut_map(func, self)


class enumerate(_coconut.enumerate):
    __slots__ = ("_iter", "_start")
    if hasattr(_coconut.enumerate, "__doc__"):
        __doc__ = _coconut.enumerate.__doc__

    def __new__(cls, iterable, start=0):
        new_enumerate = _coconut.enumerate.__new__(cls, iterable, start)
        new_enumerate._iter, new_enumerate._start = iterable, start
        return new_enumerate
```

```
    def __getitem__(self, index):
        if _coconut.isinstance(index, _coconut.slice):
            return self.__class__(
                _coconut_igetitem(self._iter, index),
                self._start
                + (
                    0
                    if index.start is None
                    else index.start
                    if index.start >= 0
                    else len(self._iter) + index.start
                ),
            )
        return (self._start + index, _coconut_igetitem(self._iter, index))

    def __len__(self):
        return _coconut.len(self._iter)

    def __repr__(self):
        return (
            "enumerate("
            + _coconut.repr(self._iter)
            + ", "
            + _coconut.repr(self._start)
            + ")"
        )

    def __reduce__(self):
        return (self.__class__, (self._iter, self._start))

    def __reduce_ex__(self, _):
        return self.__reduce__()

    def __copy__(self):
        return self.__class__(_coconut.copy.copy(self._iter), self._start)

    def __fmap__(self, func):
        return _coconut_map(func, self)


class count(object):
    """count(start, step) returns an infinite iterator starting at start and increas

    __slots__ = ("start", "step")

    def __init__(self, start=0, step=1):
        self.start, self.step = start, step

    def __iter__(self):
        while True:
            yield self.start
            self.start += self.step
```

```
def __contains__(self, elem):
    return elem >= self.start and (elem - self.start) % self.step == 0

def __getitem__(self, index):
    if (
        _coconut.isinstance(index, _coconut.slice)
        and (index.start is None or index.start >= 0)
        and (index.stop is None or index.stop >= 0)
    ):
        if index.stop is None:
            return self.__class__(
                self.start + (index.start if index.start is not None else 0),
                self.step * (index.step if index.step is not None else 1),
            )
        if _coconut.isinstance(self.start, _coconut.int) and _coconut.isinstance
            self.step, _coconut.int
        ):
            return _coconut.range(
                self.start
                + self.step * (index.start if index.start is not None else 0),
                self.start + self.step * index.stop,
                self.step * (index.step if index.step is not None else 1),
            )
        return _coconut_map(
            self.__getitem__,
            _coconut.range(
                index.start if index.start is not None else 0,
                index.stop,
                index.step if index.step is not None else 1,
            ),
        )
    if index >= 0:
        return self.start + self.step * index
    raise _coconut.IndexError("count indices must be positive")

def count(self, elem):
    """Count the number of times elem appears in the count."""
    return int(elem in self)

def index(self, elem):
    """Find the index of elem in the count."""
    if elem not in self:
        raise _coconut.ValueError(_coconut.repr(elem) + " is not in count")
    return (elem - self.start) // self.step

def __repr__(self):
    return (
        "count(" + _coconut.str(self.start) + ", " + _coconut.str(self.step) + '
    )

def __hash__(self):
```

```
        return _coconut.hash((self.start, self.step))

    def __reduce__(self):
        return (self.__class__, (self.start, self.step))

    def __copy__(self):
        return self.__class__(self.start, self.step)

    def __eq__(self, other):
        return (
            isinstance(other, self.__class__)
            and self.start == other.start
            and self.step == other.step
        )

    def __fmap__(self, func):
        return _coconut_map(func, self)


class groupsof(object):
    """groupsof(n, iterable) splits iterable into groups of size n.
    If the length of the iterable is not divisible by n, the last group may be of si

    __slots__ = ("group_size", "iter")

    def __init__(self, n, iterable):
        self.iter = iterable
        try:
            self.group_size = _coconut.int(n)
        except _coconut.ValueError:
            raise _coconut.TypeError("group size must be an int; not %r" % (n,))
        if self.group_size <= 0:
            raise _coconut.ValueError(
                "group size must be > 0; not %s" % (self.group_size,)
            )

    def __iter__(self):
        loop, iterator = True, _coconut.iter(self.iter)
        while loop:
            group = []
            for _ in _coconut.range(self.group_size):
                try:
                    group.append(_coconut.next(iterator))
                except _coconut.StopIteration:
                    loop = False
                    break
            if group:
                yield _coconut.tuple(group)

    def __len__(self):
        return _coconut.len(self.iter)
```

```
    def __repr__(self):
        return "groupsof(%r)" % (_coconut.repr(self.iter),)

    def __reduce__(self):
        return (self.__class__, (self.group_size, self.iter))

    def __copy__(self):
        return self.__class__(self.group_size, _coconut.copy.copy(self.iter))

    def __fmap__(self, func):
        return _coconut_map(func, self)


def recursive_iterator(func):
    """Decorator that optimizes a function for iterator recursion."""
    tee_store, backup_tee_store = {}, []

    @_coconut.functools.wraps(func)
    def recursive_iterator_func(*args, **kwargs):
        key, use_backup = (args, _coconut.frozenset(kwargs)), False
        try:
            hash(key)
        except _coconut.Exception:
            try:
                key = _coconut.pickle.dumps(key, _coconut.pickle.HIGHEST_PROTOCOL)
            except _coconut.Exception:
                use_backup = True
        if use_backup:
            for i, (k, v) in _coconut.enumerate(backup_tee_store):
                if k == key:
                    to_tee, store_pos = v, i
                    break
            else:  # no break
                to_tee, store_pos = func(*args, **kwargs), None
            to_store, to_return = _coconut_tee(to_tee)
            if store_pos is None:
                backup_tee_store.append([key, to_store])
            else:
                backup_tee_store[store_pos][1] = to_store
        else:
            tee_store[key], to_return = _coconut_tee(
                tee_store.get(key) or func(*args, **kwargs)
            )
        return to_return

    return recursive_iterator_func


def addpattern(base_func):
    """Decorator to add a new case to a pattern-matching function,
    where the new case is checked last."""
```

```
    def pattern_adder(func):
        @_coconut_tco
        @_coconut.functools.wraps(func)
        def add_pattern_func(*args, **kwargs):
            try:
                return base_func(*args, **kwargs)
            except _coconut_MatchError:
                return _coconut_tail_call(func, *args, **kwargs)

        return add_pattern_func

    return pattern_adder


def prepattern(base_func):
    """DEPRECATED: Use addpattern instead."""

    def pattern_prepender(func):
        return addpattern(func)(base_func)

    return pattern_prepender


class _coconut_partial(object):
    __slots__ = ("func", "_argdict", "_arglen", "_stargs", "keywords")
    if hasattr(_coconut.functools.partial, "__doc__"):
        __doc__ = _coconut.functools.partial.__doc__

    def __init__(self, func, argdict, arglen, *args, **kwargs):
        self.func, self._argdict, self._arglen, self._stargs, self.keywords = (
            func,
            argdict,
            arglen,
            args,
            kwargs,
        )

    def __reduce__(self):
        return (
            self.__class__,
            (self.func, self._argdict, self._arglen) + self._stargs,
            self.keywords,
        )

    def __setstate__(self, keywords):
        self.keywords = keywords

    @property
    def args(self):
        return (
            _coconut.tuple(self._argdict.get(i) for i in _coconut.range(self._argler
            + self._stargs
```

```
        )

    def __call__(self, *args, **kwargs):
        callargs = []
        argind = 0
        for i in _coconut.range(self._arglen):
            if i in self._argdict:
                callargs.append(self._argdict[i])
            elif argind >= _coconut.len(args):
                raise _coconut.TypeError(
                    "expected at least "
                    + _coconut.str(self._arglen - _coconut.len(self._argdict))
                    + " argument(s) to "
                    + _coconut.repr(self)
                )
            else:
                callargs.append(args[argind])
                argind += 1
        callargs += self._stargs
        callargs += args[argind:]
        kwargs.update(self.keywords)
        return self.func(*callargs, **kwargs)

    def __repr__(self):
        args = []
        for i in _coconut.range(self._arglen):
            if i in self._argdict:
                args.append(_coconut.repr(self._argdict[i]))
            else:
                args.append("?")
        for arg in self._stargs:
            args.append(_coconut.repr(arg))
        return _coconut.repr(self.func) + "$(" + ", ".join(args) + ")"


def makedata(data_type, *args, **kwargs):
    """Call the original constructor of the given data type or class with the given
    if _coconut.hasattr(data_type, "_make") and (
        _coconut.issubclass(data_type, _coconut.tuple)
        or _coconut.isinstance(data_type, _coconut.tuple)
    ):
        return data_type._make(args, **kwargs)
    return _coconut.super(data_type, data_type).__new__(data_type, *args, **kwargs)


def datamaker(data_type):
    """DEPRECATED: Use makedata instead."""
    return _coconut.functools.partial(makedata, data_type)


def consume(iterable, keep_last=0):
    """consume(iterable, keep_last) fully exhausts iterable and return the last keep
```

```
    return _coconut.collections.deque(
        iterable, maxlen=keep_last
    )  # fastest way to exhaust an iterator


class starmap(_coconut.itertools.starmap):
    __slots__ = ("_func", "_iter")
    if hasattr(_coconut.itertools.starmap, "__doc__"):
        __doc__ = _coconut.itertools.starmap.__doc__

    def __new__(cls, function, iterable):
        new_map = _coconut.itertools.starmap.__new__(cls, function, iterable)
        new_map._func, new_map._iter = function, iterable
        return new_map

    def __getitem__(self, index):
        if _coconut.isinstance(index, _coconut.slice):
            return self.__class__(self._func, _coconut_igetitem(self._iter, index))
        return self._func(*_coconut_igetitem(self._iter, index))

    def __reversed__(self):
        return self.__class__(self._func, *_coconut_reversed(self._iter))

    def __len__(self):
        return _coconut.len(self._iter)

    def __repr__(self):
        return (
            "starmap("
            + _coconut.repr(self._func)
            + ", "
            + _coconut.repr(self._iter)
            + ")"
        )

    def __reduce__(self):
        return (self.__class__, (self._func, self._iter))

    def __reduce_ex__(self, _):
        return self.__reduce__()

    def __copy__(self):
        return self.__class__(self._func, _coconut.copy.copy(self._iter))

    def __fmap__(self, func):
        return self.__class__(_coconut_forward_compose(self._func, func), self._iter


def fmap(func, obj):
    """fmap(func, obj) creates a copy of obj with func applied to its contents.
    Override by defining .__fmap__(func)."""
    if _coconut.hasattr(obj, "__fmap__"):
```

```
        return obj.__fmap__(func)
    args = (
        _coconut_starmap(func, obj.items())
        if _coconut.isinstance(obj, _coconut.abc.Mapping)
        else _coconut_map(func, obj)
    )
    if _coconut.isinstance(obj, _coconut.tuple) and _coconut.hasattr(obj, "_make"):
        return obj._make(args)
    if _coconut.isinstance(obj, (_coconut.map, _coconut.range, _coconut.abc.Iterator
        return args
    if _coconut.isinstance(obj, _coconut.str):
        return "".join(args)
    return obj.__class__(args)


(
    _coconut_MatchError,
    _coconut_count,
    _coconut_enumerate,
    _coconut_reversed,
    _coconut_map,
    _coconut_starmap,
    _coconut_tee,
    _coconut_zip,
    reduce,
    takewhile,
    dropwhile,
) = (
    MatchError,
    count,
    enumerate,
    reversed,
    map,
    starmap,
    tee,
    zip,
    _coconut.functools.reduce,
    _coconut.itertools.takewhile,
    _coconut.itertools.dropwhile,
)

# Compiled Coconut: ------------------------------------------------------------


def factorial_variant_A(n):
    _coconut_match_to = n
    _coconut_match_check = False
    if _coconut_match_to == 0:
        _coconut_match_check = True
    if _coconut_match_check:
        return 1
    if not _coconut_match_check:
```

```
        if _coconut_match_to == 1:
            _coconut_match_check = True
        if _coconut_match_check:
            return 1
    if not _coconut_match_check:
        x = _coconut_match_to
        _coconut_match_check = True
        if _coconut_match_check:
            return x * factorial_variant_A(x - 1)
    if not _coconut_match_check:
        raise TypeError("expecting integer >= 0")


for n in range(11):
    print("{n}!={f}".format(n=n, f=factorial_variant_A(n)))

# !!!!!!! print(factorial_variant_A(-10))


def factorial_variant_B(n):
    _coconut_match_to = n
    _coconut_match_check = False
    if _coconut_match_to == 0:
        _coconut_match_check = True
    if _coconut_match_check:
        return 1
    if not _coconut_match_check:
        if _coconut_match_to == 1:
            _coconut_match_check = True
        if _coconut_match_check:
            return 1
    if not _coconut_match_check:
        x = _coconut_match_to
        _coconut_match_check = True
        if _coconut_match_check and not (x > 1):
            _coconut_match_check = False
        if _coconut_match_check:
            return x * factorial_variant_B(x - 1)
    if not _coconut_match_check:
        raise TypeError("expecting integer >= 0")


for n in range(11):
    print("{n}!={f}".format(n=n, f=factorial_variant_B(n)))

# print(factorial_variant_B(1.2))
# print(factorial_variant_B(-10))


def factorial_variant_C(n):
    _coconut_match_to = n
    _coconut_match_check = False
```

```
    if _coconut_match_to == 0:
        _coconut_match_check = True
    if _coconut_match_check:
        return 1
    if not _coconut_match_check:
        if _coconut_match_to == 1:
            _coconut_match_check = True
        if _coconut_match_check:
            return 1
    if not _coconut_match_check:
        if _coconut.isinstance(_coconut_match_to, int):
            x = _coconut_match_to
            _coconut_match_check = True
        if _coconut_match_check and not (x > 1):
            _coconut_match_check = False
        if _coconut_match_check:
            return x * factorial_variant_C(x - 1)
    if not _coconut_match_check:
        raise TypeError("expecting integer >= 0")


for n in range(11):
    print("{n}!={f}".format(n=n, f=factorial_variant_C(n)))


def factorial_variant_D(n):
    _coconut_match_to = n
    _coconut_match_check = False
    if _coconut_match_to == 0:
        _coconut_match_check = True
    if _coconut_match_check:
        return 1
    if not _coconut_match_check:
        if _coconut_match_to == 1:
            _coconut_match_check = True
        if _coconut_match_check:
            return 1
    if not _coconut_match_check:
        if _coconut.isinstance(_coconut_match_to, int):
            _coconut_match_check = True
        if _coconut_match_check and not (n > 1):
            _coconut_match_check = False
        if _coconut_match_check:
            return n * factorial_variant_D(n - 1)
    if not _coconut_match_check:
        raise TypeError("expecting integer >= 0")


for n in range(11):
    print("{n}!={f}".format(n=n, f=factorial_variant_D(n)))
```

Zdrojový kód příkladu

Pattern matching

```
def type(x):
    case x:
        match [a, b]:
            return "list, 2 items"
        match [a]:
            return "list, 1 item"
        match []:
            return "empty list"

def say_hello(s):
    case s:
        match "My name is " + name:
            return "Hi " + name

def pair(p):
    case p:
        match [x,x]:
            return "same values!"
        match [x,y] if x>y:
            return "1st value is greater"
        match [x,y]:
            return "2nd value is greater"
    else:
        return "other"

def get_name(s):
    case s:
        match name + "@root.cz":
            return name
```

Zdrojový kód příkladu

Pattern matching

```
def type(x):
    _coconut_match_to = x
    _coconut_match_check = False
    if (_coconut.isinstance(_coconut_match_to, _coconut.abc.Sequence)) and (_coconut
        a = _coconut_match_to[0]
        b = _coconut_match_to[1]
```

```
        _coconut_match_check = True
    if _coconut_match_check:
        return "list, 2 items"
    if not _coconut_match_check:
        if (_coconut.isinstance(_coconut_match_to, _coconut.abc.Sequence)) and (_coc
            a = _coconut_match_to[0]
            _coconut_match_check = True
        if _coconut_match_check:
            return "list, 1 item"
    if not _coconut_match_check:
        if (_coconut.isinstance(_coconut_match_to, _coconut.abc.Sequence)) and (_coc
            _coconut_match_check = True
        if _coconut_match_check:
            return "empty list"

def say_hello(s):
    _coconut_match_to = s
    _coconut_match_check = False
    if (_coconut.isinstance(_coconut_match_to, _coconut.str)) and (_coconut_match_to
        name = _coconut_match_to[_coconut.len("My name is "):]
        _coconut_match_check = True
    if _coconut_match_check:
        return "Hi " + name

def pair(p):
    _coconut_match_to = p
    _coconut_match_check = False
    if (_coconut.isinstance(_coconut_match_to, _coconut.abc.Sequence)) and (_coconut
        x = _coconut_match_to[0]
        _coconut_match_check = True
    if _coconut_match_check:
        return "same values!"
    if not _coconut_match_check:
        if (_coconut.isinstance(_coconut_match_to, _coconut.abc.Sequence)) and (_coc
            x = _coconut_match_to[0]
            y = _coconut_match_to[1]
            _coconut_match_check = True
        if _coconut_match_check and not (x > y):
            _coconut_match_check = False
        if _coconut_match_check:
            return "1st value is greater"
    if not _coconut_match_check:
        if (_coconut.isinstance(_coconut_match_to, _coconut.abc.Sequence)) and (_coc
            x = _coconut_match_to[0]
            y = _coconut_match_to[1]
            _coconut_match_check = True
        if _coconut_match_check:
            return "2nd value is greater"
    if not _coconut_match_check:
        return "other"

def get_name(s):
```

```
    _coconut_match_to = s
    _coconut_match_check = False
    if (_coconut.isinstance(_coconut_match_to, _coconut.str)) and (_coconut_match_tc
        name = _coconut_match_to[:-_coconut.len("@root.cz")]
        _coconut_match_check = True
    if _coconut_match_check:
        return name
```

Zdrojový kód příkladu

TCO

```
def factorial_tco(n, acc=1):
    case n:
        match 0:
            return acc
        match 1:
            return acc
        match _ is int if n > 1:
            return factorial_tco(n-1, acc*n)
    else:
        raise TypeError("expecting integer >= 0")

for n in range(11):
    print("{n}!={f}".format(n=n, f=factorial_tco(n)))

print(factorial_tco(1000))
print(factorial_tco(10000))
```

Zdrojový kód příkladu

TCO

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# __coconut_hash__ = 0x8f8983fe

# Compiled with Coconut version 1.3.0 [Dead Parrot]

# Coconut Header: --------------------------------------------------------------


# Compiled Coconut: ------------------------------------------------------------
```

```python
@_coconut_tco
def factorial_tco(n, acc=1):
    def _coconut_mock_func(n, acc=1):
        return n, acc

    while True:
        _coconut_match_to = n
        _coconut_match_check = False
        if _coconut_match_to == 0:
            _coconut_match_check = True
        if _coconut_match_check:
            return acc
        if not _coconut_match_check:
            if _coconut_match_to == 1:
                _coconut_match_check = True
            if _coconut_match_check:
                return acc
        if not _coconut_match_check:
            if _coconut.isinstance(_coconut_match_to, int):
                _coconut_match_check = True
            if _coconut_match_check and not (n > 1):
                _coconut_match_check = False
            if _coconut_match_check:
                if factorial_tco is _coconut_recursive_func_0:
                    n, acc = _coconut_mock_func(n - 1, acc * n)
                    continue
                else:
                    return _coconut_tail_call(factorial_tco, n - 1, acc * n)
        if not _coconut_match_check:
            raise TypeError("expecting integer >= 0")

        return None


_coconut_recursive_func_0 = factorial_tco
for n in range(11):
    print("{n}!={f}".format(n=n, f=factorial_tco(n)))

print(factorial_tco(1000))
print(factorial_tco(10000))
```

Zdrojový kód příkladu

Mojo

Testování

Python

- Základní technologie testování
- Pyramida testů
- Zmrzlinový kornout jako antipattern
- Jednotkové testy
- Modul `pytest`
- Nástroj Hypothesis
- Fuzzy testy

Testovací frameworky v Pythonu

```
1                   unittest
2                   doctest
3                   pytest
4                   nose
5                   testify
6                   Trial
7                   Twisted
8                   subunit
9                   testresources
10                  reahl.tofu
11                  unit testing
12                  testtools
13                  Sancho
14                  zope.testing
15                  pry
16                  pythoscope
17                  testlib
18                  pytest
19                  dutest
```

Pyramida typů testů

- Business část
  - Beta testy
  - Alfa testy
  - Akceptační testy

- Technologická část
  - UI testy
  - API testy
  - Integrační testy
  - Testy komponent
  - Unit testy
- Další typy testů
  - Benchmarky