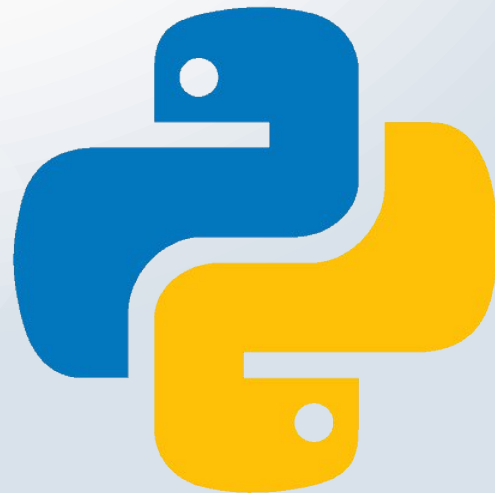


hardcore_python() ◀

Mikołaj Storoniak

Poznaj węża

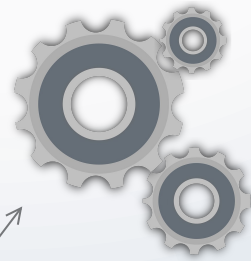
- Python to język programowania
 - Interpretowany
 - Wysokiego poziomu
 - Dynamicznie typowany
 - Z automatycznym zarządzaniem pamięcią



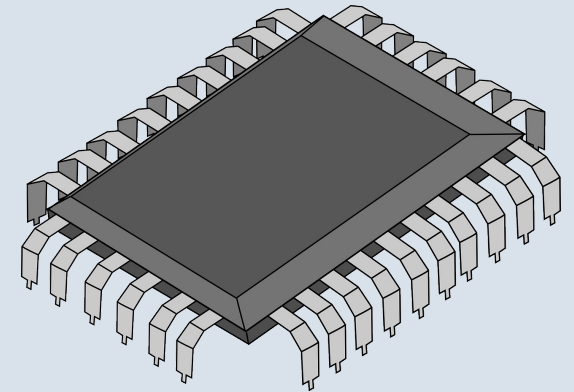
Interpretacja? Kompilacja?

- Języki kompilowane („tradycyjne”):
 1. Kod
 2. Kompilacja
 3. Kod maszynowy (binarny)
 4. Wykonanie

```
1  /* C basic structure explained
2     Author: fresh2refresh.com // Documentation section
3     Date   : 01/01/2012
4  */
5  #include <stdio.h>           // Link section
6  int total;                   // Global declaration section
7  int sum (int, int);          // Function declaration section
8  int main ()                  // Main function
9  {
10     printf ("This is main function \n");
11     total = sum (1, 1);
12     printf ("Total = %d \n", total);
13     return 0;
14 }
15 int sum (int a, int b)
16 {
17     return a + b;             // User defined function
18                               // definition section
19 }
```



```
0: eb 63      jmp     0x65
2: 90          nop
3: 10 8e d0 bc  adc     BYTE PTR [bp-0x4330],cl
7: 00 b0 b8 00  add     BYTE PTR [bx+si+0xb8],dh
b: 00 8e d8 8e  add     BYTE PTR [bp-0x7128],cl
f: c0 fb be    sar     bl,0xbe
12: 00 7c bf     add     BYTE PTR [si-0x41],bh
15: 00 06 b9 00  add     BYTE PTR ds:0xb9,a1
19: 02 f3       add     dh,bl
1b: a4         movs    BYTE PTR es:[di],BYTE PTR ds:[si]
1c: ea 21 06 00 00 jmp     0x0:0x621
21: be be 07     mov     si,0x7be
24: 38 04       cmp     BYTE PTR [si],a1
26: 75 0b       jne     0x33
28: 83 c6 10     add     si,0x10
2b: 81 fe fe 07  cmp     si,0x7fe
2f: 75 f3       jne     0x24
31: eb 16       jmp     0x49
33: b4 02       mov     ah,0x2
35: b0 01       mov     al,0x1
37: bb 00 7c    mov     bx,0x7c00
3a: b2 80       mov     dl,0x80
3c: 8a 74 01     mov     dh,BYTE PTR [si+0x1]
3f: 8b 4c 02     mov     cx,WORD PTR [si+0x2]
42: cd 13       int     0x13
44: ea 00 7c 00 00 jmp     0x0:0x7c00
```



Interpretacja? Kompilacja?

- Języki interpretowane
 1. Kod
 2. Interpretacja „linijka po linijce” (interpreter)
 3. Kompilacja (opcjonalnie)
 4. Wykonanie



Interpreter Pythona

1. Analiza kodu - Sprawdzenie składni, podział na tokeny
 2. Generacja Bytecode (pycache)
 3. Python Virtual Machine
- Istnieje wiele interpreterów:
<https://hackr.io/blog/python-interpreters>
 - CPython jest uznawany za „domyślny”

Przykład

python -m dis nazwa.py

```
0      0 RESUME                0
1      2 LOAD_CONST             0 (<code object hardcore_python_function at 0x7f6243fe72f0, file "sample.py", line 1>)
      4 MAKE_FUNCTION
      6 STORE_NAME               0 (hardcore_python_function)

5      8 LOAD_CONST             1 (<code object even_more_hardware at 0x7f6243e9c030, file "sample.py", line 5>)
     10 MAKE_FUNCTION
     12 STORE_NAME               1 (even_more_hardware)

12     14 PUSH_NULL
     16 LOAD_NAME                0 (hardcore_python_function)
     18 PRECALL
     22 CALL
     32 POP_TOP

14     34 PUSH_NULL
     36 LOAD_NAME                1 (even_more_hardware)
     38 PRECALL
     42 CALL
     52 POP_TOP
     54 LOAD_CONST              2 (None)
     56 RETURN_VALUE

Disassembly of <code object hardcore_python_function at 0x7f6243fe72f0, file "sample.py", line 1>:
1      0 RESUME                0

2      2 LOAD_GLOBAL              1 (NULL + print)
     14 LOAD_CONST              1 ('wąż rzeczny')
     16 PRECALL
     20 CALL
     30 POP_TOP

3     32 LOAD_CONST              2 (1337)
     34 RETURN_VALUE
```

.....
.....
.....

```
1  def hardcore_python_function():
2      print("wąż rzeczny")
3      return 1337
4
5  def even_more_hardware():
6      for i in range(10):
7          print(i*i)
8      return 0
9
10
11
12  hardcore_python_function()
13
14  even_more_hardware()
```

Bytecode

```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Decoded Text
00000000 A7 0D 0D 0A 00 00 00 00 D9 E5 55 65 D2 00 00 00 U e
00000010 E3 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00
00000020 00 00 00 00 00 F3 3A 00 00 00 97 00 64 00 84 00
00000030 5A 00 64 01 84 00 5A 01 02 00 65 00 A6 00 00 00 Z d
00000040 AB 00 00 00 00 00 00 00 00 00 00 01 00 02 00 65 01
00000050 A6 00 00 00 AB 00 00 00 00 00 00 00 00 00 01 00
00000060 64 02 53 00 29 03 63 00 00 00 00 00 00 00 00 00 d S ) c
00000070 00 00 00 03 00 00 00 03 00 00 00 F3 24 00 00 00 $
00000080 97 00 74 01 00 00 00 00 00 00 00 00 00 64 01 t
00000090 A6 01 00 00 AB 01 00 00 00 00 00 00 00 00 01 00
000000A0 64 02 53 00 29 03 4E 75 0D 00 00 00 77 C4 85 C5 d S ) N u
000000B0 BC 20 72 7A 65 63 7A 6E 79 69 39 05 00 00 29 01 r z e c z n y i 9
000000C0 DA 05 70 72 69 6E 74 A9 00 F3 00 00 00 00 FA 09 p r i n t
000000D0 73 61 6D 70 6C 65 2E 70 79 DA 18 68 61 72 64 63 s a m p l e . p y h a r d c
000000E0 6F 72 65 5F 70 79 74 68 6F 6E 5F 66 75 6E 63 74 o r e _ p y t h o n _ f u n c t
000000F0 69 6F 6E 72 07 00 00 00 01 00 00 00 73 15 00 00 i o n r
00000100 00 80 00 DD 04 09 88 2F D1 04 1A D4 04 1A D0 04
00000110 1A D8 0B 0F 88 34 72 05 00 00 00 63 00 00 00
00000120 00 00 00 00 00 00 00 00 05 00 00 00 03 00 00 00
00000130 F3 4E 00 00 00 97 00 74 01 00 00 00 00 00 00 00
00000140 00 00 00 64 01 A6 01 00 00 AB 01 00 00 00 00 00
00000150 00 00 00 44 00 5D 14 7D 00 74 03 00 00 00 00 00
00000160 00 00 00 00 00 7C 00 7C 00 7A 05 00 00 A6 01 00
00000170 00 AB 01 00 00 00 00 00 00 00 00 01 00 8C 15 64
00000180 02 53 00 29 03 4E E9 0A 00 00 00 E9 00 00 00 00
00000190 29 02 DA 05 72 61 6E 67 65 72 03 00 00 00 29 01
000001A0 DA 01 69 73 01 00 00 00 20 72 06 00 00 00 DA 12
000001B0 65 76 65 6E 5F 6D 6F 72 65 5F 68 61 72 64 63 6F
000001C0 72 65 72 0D 00 00 00 05 00 00 00 73 2D 00 00 00
000001D0 80 00 DD 0D 12 90 32 89 59 8C 59 F0 00 01 05 13
000001E0 F0 00 01 05 13 88 01 DD 08 0D 88 61 90 01 89 63
000001F0 89 0A 8C 0A 88 0A 88 0A D8 0B 0C 88 31 72 05 00
00000200 00 00 4E 29 02 72 07 00 00 00 72 0D 00 00 00 72
00000210 04 00 00 00 72 05 00 00 00 72 06 00 00 00 FA 08
00000220 3C 6D 6F 64 75 6C 65 3E 72 0E 00 00 00 01 00 00
00000230 00 73 49 00 00 00 F0 03 01 01 01 F0 02 02 01 10
00000240 F0 00 02 01 10 F0 00 02 01 10 F0 08 03 01 0D F0
00000250 00 03 01 0D F0 00 03 01 0D F0 0E 00 01 19 D0 00
00000260 18 D1 00 1A D4 00 1A D0 00 1A E0 00 12 D0 00 12
00000270 D1 00 14 D4 00 14 D0 00 14 D0 00 14 D0 00 14 72
00000280 05 00 00 00
```

```
python -m py_compile nazwa.py
```


Bytecode

```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Decoded Text
00000000 A7 0D 0D 0A 00 00 00 00 D9 E5 55 65 D2 00 00 00 U e
00000010 E3 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00
00000020 00 00 00 00 00 F3 3A 00 00 00 97 00 64 00 84 00
00000030 5A 00 64 01 84 00 5A 01 02 00 65 00 A6 00 00 00 Z d
00000040 AB 00 00 00 00 00 00 00 00 00 01 00 02 00 65 01 e
00000050 A6 00 00 00 AB 00 00 00 00 00 00 00 00 00 01 00
00000060 64 02 53 00 29 03 63 00 00 00 00 00 00 00 00 d S ) c
00000070 00 00 00 03 00 00 00 03 00 00 00 F3 24 00 00 00 $
00000080 97 00 74 01 00 00 00 00 00 00 00 00 00 64 01 t
00000090 A6 01 00 00 AB 01 00 00 00 00 00 00 00 00 01 00
000000A0 64 02 53 00 29 03 E 75 0D 00 00 00 77 C4 85 C5 d S ) N u
000000B0 BC 20 72 7A 65 63 7A 6E 79 69 39 05 00 00 29 01 r z e c z n y i 9
000000C0 DA 05 70 72 69 6E 74 A9 00 F3 00 00 00 00 FA 09 print
000000D0 73 61 6D 70 6C 65 2E 70 79 DA 18 68 61 72 64 63 sample.py hardc
000000E0 6F 72 65 5F 70 79 74 68 6F 6E 5F 66 75 6E 63 74 ore_python_funct
000000F0 69 6F 6E 72 07 00 00 00 01 00 00 00 73 15 00 00 ionr
00000100 00 80 00 DD 04 09 88 2F D1 04 1A D4 04 1A D0 04
00000110 1A D8 0B 0F 88 34 72 05 00 00 00 63 00 00 00
00000120 00 00 00 00 00 00 00 00 05 00 00 00 03 00 00 00
00000130 F3 4E 00 00 00 97 00 74 01 00 00 00 00 00 00 00
00000140 00 00 00 64 01 A6 01 00 00 AB 01 00 00 00 00 00
00000150 00 00 00 44 00 5D 14 7D 00 74 03 00 00 00 00 00
00000160 00 00 00 00 00 7C 00 7C 00 7A 05 00 00 A6 01 00
00000170 00 AB 01 00 00 00 00 00 00 00 00 01 00 8C 15 64
00000180 02 53 00 29 03 4E E9 0A 00 00 00 E9 00 00 00 00
00000190 29 02 DA 05 72 61 6E 67 65 72 03 00 00 00 29 01
000001A0 DA 01 69 73 01 00 00 00 20 72 06 00 00 00 DA 12
000001B0 65 76 65 6E 5F 6D 6F 72 65 5F 68 61 72 64 63 6F
000001C0 72 65 72 0D 00 00 00 05 00 00 00 73 2D 00 00 00
000001D0 80 00 DD 0D 12 90 32 89 59 8C 59 F0 00 01 05 13
000001E0 F0 00 01 05 13 88 01 DD 08 0D 88 61 90 01 89 63
000001F0 89 0A 8C 0A 88 0A 88 0A D8 0B 0C 88 31 72 05 00
00000200 00 00 4E 29 02 72 07 00 00 00 72 0D 00 00 00 72
00000210 04 00 00 00 72 05 00 00 00 72 06 00 00 00 FA 08
00000220 3C 6D 6F 64 75 6C 65 3E 72 0E 00 00 00 01 00 00
00000230 00 73 49 00 00 00 F0 03 01 01 01 F0 02 02 01 10
00000240 F0 00 02 01 10 F0 00 02 01 10 F0 08 03 01 0D F0
00000250 00 03 01 0D F0 00 03 01 0D F0 0E 00 01 19 D0 00
00000260 18 D1 00 1A D4 00 1A D0 00 1A E0 00 12 D0 00 12
00000270 D1 00 14 D4 00 14 D0 00 14 D0 00 14 D0 00 14 72
00000280 05 00 00 00
```

`python -m py_compile nazwa.py`

- Zadanie: napisać przykładowy program i sprawdzić wynik kompilacji
- Warto sprawdzić różne typy danych, funkcje wbudowane, deklaracje klasy...

Wszystko jest obiektem

- Każdy fragment kodu, funkcja i ramka stosu to obiekt

```
struct PyCodeObject {
    PyObject_HEAD
    int co_argcount;           /* #arguments, except *args */
    int co_posonlyargcount;    /* #positional only arguments */
    int co_kwonlyargcount;    /* #keyword only arguments */
    int co_nlocals;           /* #local variables */
    int co_stacksize;         /* #entries needed for evaluation stack */
    int co_flags;             /* CO_..., see below */
    int co_firstlineno;       /* first source line number */
    PyObject *co_code;        /* instruction opcodes */
    PyObject *co_consts;      /* list (constants used) */
    PyObject *co_names;       /* list of strings (names used) */
    PyObject *co_varnames;    /* tuple of strings (local variable names) */
    PyObject *co_freevars;    /* tuple of strings (free variable names) */
    PyObject *co_cellvars;    /* tuple of strings (cell variable names) */

    Py_ssize_t *co_cell2arg;   /* Maps cell vars which are arguments. */
    PyObject *co_filename;    /* unicode (where it was loaded from) */
    PyObject *co_name;        /* unicode (name, for reference) */
    /* ... more members ... */
};
```

Wszystko jest obiektem

- Każdy fragment kodu, funkcja i ramka stosu to obiekt

```
typedef struct {
    PyObject_HEAD
    PyObject *func_code;           /* A code object, the __code__ attribute */
    PyObject *func_globals;        /* A dictionary (other mappings won't do) */
    PyObject *func_defaults;       /* NULL or a tuple */
    PyObject *func_kwdefaults;     /* NULL or a dict */
    PyObject *func_closure;        /* NULL or a tuple of cell objects */
    PyObject *func_doc;            /* The __doc__ attribute, can be anything */
    PyObject *func_name;           /* The __name__ attribute, a string object */
    PyObject *func_dict;           /* The __dict__ attribute, a dict or NULL */
    PyObject *func_weakreflist;    /* List of weak references */
    PyObject *func_module;         /* The __module__ attribute, can be anything */
    PyObject *func_annotations;    /* Annotations, a dict or NULL */
    PyObject *func_qualname;       /* The qualified name */
    vectorcallfunc vectorcall;
} PyFunctionObject;
```

Wszystko jest obiektem

- Każdy fragment kodu, funkcja i ramka stosu to obiekt

```
struct _frame {
    PyObject_VAR_HEAD
    struct _frame *f_back;      /* previous frame, or NULL */
    PyCodeObject *f_code;       /* code segment */
    PyObject *f_builtins;       /* builtin symbol table (PyDictObject) */
    PyObject *f_globals;        /* global symbol table (PyDictObject) */
    PyObject *f_locals;         /* local symbol table (any mapping) */
    PyObject **f_valuelist;      /* points after the last local */

    PyObject **f_stacktop;       /* Next free slot in f_valuelist. ... */
    PyObject *f_trace;          /* Trace function */
    char f_trace_lines;         /* Emit per-line trace events? */
    char f_trace_opcodes;       /* Emit per-opcode trace events? */

    /* Borrowed reference to a generator, or NULL */
    PyObject *f_gen;

    int f_lasti;                /* Last instruction if called */
    /* ... */
    int f_lineno;               /* Current line number */
    int f_iblock;               /* index in f_blockstack */
    char f_executing;           /* whether the frame is still executing */
    PyTryBlock f_blockstack[CO_MAXBLOCKS]; /* for try and loop blocks */
    PyObject *f_localsplus[1]; /* locals+stack, dynamically sized */
};
```

Tablica nazw

- Python rozpoznaje zmienne po nazwie (nie po adresie)
- Można odwoływać się do nieistniejących atrybutów (tylko przy zapisie)

```
def hardcore_python_function():  
    y = 20  
    print("wąż rzeczny")  
    return 1337  
  
hardcore_python_function.x = 10  
  
print(hardcore_python_function.x)  
#To nie zadziała  
#print(hardcore_python_function.z)  
#print(hardcore_python_function.y)
```

```
def hardcore_python_function():  
    y = 20  
    print("wąż rzeczny")  
    return 1337  
  
hardcore_python_function.x = 10  
snek = "python"  
print(snek)  
#print(snek())  
snek = hardcore_python_function  
  
print(hardcore_python_function.x)  
print(snek())  
print(snek.x)
```

Stos wywołań

- Czy stos wywołań ma nieograniczoną pojemność?



Stos wywołań

- Czy stos wywołań ma nieograniczoną pojemność?
- Zadanie: Sprawdzić jego pojemność

Stos wywołań

```
def epic_recursion(x):  
    print(x)  
    epic_recursion(x+1)  
  
epic_recursion(1)
```

```
epic_recursion(x+1)  
File "/home/mikostoro/Documents/PG_Lore_V/Hardcore_Python/recursion.py", line 3, in epic_recursion  
    epic_recursion(x+1)  
[Previous line repeated 993 more times]  
File "/home/mikostoro/Documents/PG_Lore_V/Hardcore_Python/recursion.py", line 2, in epic_recursion  
    print(x)  
RecursionError: maximum recursion depth exceeded while calling a Python object
```

```
sys.setrecursionlimit(1500)
```

Wyjątki

- Wyjątek służy do ratowania sytuacji, kiedy coś pójdzie źle
- Do obsługi błędów służy konstrukcja try, except, finally
- Możemy definiować własne wyjątki
- Po wystąpieniu, wyjątek „wędruje do góry”, dopóki nie zostanie obsłużony
- Nieobsłużony wyjątek wysypuje program (ale zwykle zapewnia użytkownikowi istotne dane)

Wyjątki: przykład

Definicja wyjątku

```
class ZeroDivision(BaseException):  
    pass  
  
class NotInt(BaseException):  
    def __init__(self, value="value"):  
        print("ERROR: " + str(value) + " is not a number")  
        super().__init__()
```

Obsługa wyjątku

```
def epicFunction(x,y):  
    return divide(x,y)  
  
print(epicFunction(1,2))  
try:  
    print(divide("1",2))  
except BaseException:  
    pass  
try:  
    print(epicFunction(2,0))  
except NotInt:  
    pass
```

Wystąpienie wyjątku

```
def divide(dzielna, dzielnik):  
    if not isinstance(dzielna, int):  
        raise NotInt(dzielna)  
    if not isinstance(dzielnik, int):  
        raise NotInt(dzielnik)  
    if dzielnik == 0:  
        raise ZeroDivision()  
    return dzielna/dzielnik  
  
def epicFunction(x,y):  
    return divide(x,y)
```

Wyjątki: przykład

Definicja wyjątku

```
class ZeroDivision(BaseException):  
    pass  
  
class NotInt(BaseException):  
    def __init__(self, value="value"):  
        print("ERROR: " + str(value) + " is not a number")  
        super().__init__()
```

Obsługa wyjątku

```
def epicFunction(x,y):  
    return divide(x,y)  
  
print(epicFunction(1,2))  
try:  
    print(divide("1",2))  
except BaseException:  
    pass  
try:  
    print(epicFunction(2,0))  
except NotInt:  
    pass
```

Wystąpienie wyjątku

```
def divide(dzielna, dzielnik):  
    if not isinstance(dzielna, int):  
        raise NotInt(dzielna)  
    if not isinstance(dzielnik, int):  
        raise NotInt(dzielnik)  
    if dzielnik == 0:  
        raise ZeroDivision()  
    return dzielna/dzielnik  
  
def epicFunction(x,y):  
    return divide(x,y)
```

Zadanie: Napisz funkcję, która przyjmuje str i wyświetla go. Jeśli string ma powyżej 10 znaków, ma wystąpić wyjątek. W dowolny sposób obsłuż wyjątek (w bloku except)

Wbudowane wyjątki

- **IndexError** – odwołujemy się do złego indeksu
- **KeyboardInterrupt** – przerwanie z klawiatury (ctrl+c)
- **KeyError** – w słowniku nie ma klucza
- **NameError** – nie odnaleziono nazwy

...oraz wiele innych

Pełna lista:

<https://docs.python.org/3/library/exceptions.html#BaseException>

Moduły

- Dowolny plik pythonowy może zostać zaimportowany jako moduł (mamy wtedy dostęp do jego treści)
- Używamy nazwy bez `.py`
- **python -m** służy do uruchomienia zainstalowanego modułu
- Moduły mogą być też pisane w C
- Moduł „wykonuje się” w chwili zaimportowania (automatycznie wykonuje się kod poza funkcjami)
- W naszym kodzie, możemy nazwać moduł
- Możemy zaimportować część modułu

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
```

```
from numpy import array as nparray

arr = nparray([1, 2, 3, 4, 5])
```


Moduły

- Moduł można wykonać jako skrypt
- Jeśli mamy kod poza funkcjami, to odpali się też przy imporcie

```
def adder(a, b):  
    return a + b  
  
def subtractor(a,b):  
    return a-b  
  
print("module imported")
```

```
>>> import moduleExample  
module imported  
>>>
```

- Co robić? Jak żyć?

Moduły

- Python zna nazwy swoich funkcji i zmiennych (w odróżnieniu od języków niższego poziomu)
- Funkcja `__main__()` jest pierwszą funkcją, która wywołuje się po uruchomieniu interpretera
- Sprawdźmy zatem, czy jesteśmy w funkcji `__main__()`

Moduły

- Python zna nazwy swoich funkcji i zmiennych (w odróżnieniu od języków niższego poziomu)
- Funkcja `__main__()` jest pierwszą funkcją, która wywołuje się po uruchomieniu interpretera
- Sprawdźmy zatem, czy jesteśmy w funkcji `__main__()`

```
if __name__ == "__main__":  
    print("module imported")
```

```
>>> import moduleExample  
module imported  
>>> import moduleExample2  
>>>
```

```
[mikostoro@billhates Hardcore_Python]$ python -m moduleExample2.py  
/usr/bin/python: Error while finding module specification for 'moduleExample2.py' (ModuleNotFoundError)  
[mikostoro@billhates Hardcore_Python]$ python -m moduleExample2  
module imported
```

Zalety i wady Pythona

Zalety i wady Pythona

- + Przenośność
 - + Zarządzanie pamięcią
 - + Łatwość użycia
 - + Dynamiczne typowanie
 - + Odwołania po nazwach
 - + Odporność na błędy (wyjątki, segmentation fault)
-
- Prędkość
 - Złe nawyki
 - Wymagania sprzętowe
 - Podatność na błędy (dynamiczne typowanie)

Tryb interaktywny

- Uruchomienie:
`python`
- Ostatnia uzyskana wartość: `_`
- Wprowadzenie nazwy zmiennej powoduje jej wyświetlenie
- Wywołanie funkcji wypisuje wynik
- Jeśli funkcja zwraca `None`, wynik nie jest wyświetlany
`print (print ("text"))`
- Symbol „>>>” oznacza początek komendy.
- Symbol „...” oznacza ciąg dalszy komendy (dalej trzeba pamiętać o indentacji)

Środowiska wirtualne

- Środowisko do uruchamiania skryptów
- Jest dołączone do konkretnego interpretera
- Zawiera zbiór bibliotek
- Łatwe do utworzenia
- Zależne od urządzenia (nieprzenośne)

Środowiska wirtualne - zalety

Środowiska wirtualne - zalety

- Wiele wersji danej biblioteki
- Biblioteki potrzebne dla jednego projektu
- Porządek w plikach
- Łatwa konfiguracja środowiska
- Unikanie konfliktów
- Wybór interpretera

Środowiska wirtualne - wady

Środowiska wirtualne - wady

Nie

Praca ze środowiskami wirtualnymi

- **Utworzenie środowiska:**

```
python -m venv /path/to/venv
```

- **Pomoc:**

```
python -h venv
```

- **Aktywacja:**

```
source path/to/venv/bin/activate
```

- **Dezaktywacja:**

```
deactivate
```


Praca ze środowiskami wirtualnymi

- Utworzenie środowiska:
`python -m venv /path/to/venv`
- Pomoc:
`python -h venv`
- Aktywacja:
`source path/to/venv/bin/activate`
- Dezaktywacja:
`deactivate`
- Zadanie: Przygotuj środowisko wirtualne i zainstaluj bibliotekę **bottle**

Przygotowanie środowiska

- Mając aktywne środowisko, możemy instalować w nim pakiety

```
python -m pip install 'nazwa_pakietu'
```

- Pakiety zostaną zainstalowane tylko w tym środowisku
- Zadanie: Sprawdzić czy dostępne są pakiety systemowe?

Przygotowanie środowiska

- Mając aktywne środowisko, możemy instalować w nim pakiety

```
python -m pip install 'nazwa_pakietu'
```

- Pakiety zostaną zainstalowane tylko w tym środowisku
- Zadanie: Sprawdzić czy dostępne są pakiety systemowe?

```
--system-site-packages
```

Czy musimy wszystko instalować ręcznie?

Czy musimy wszystko instalować ręcznie?

- Plik z wymaganiami:

```
python -m pip freeze >requirements.txt
```

- Instalacja z pliku z wymaganiami:

```
Python -m pip install -r requirements.txt
```

- Nie należy zapominać o uprzednim uruchomieniu środowiska wirtualnego!

Źródła

- <https://docs.python.org/3/>
- <https://totalbooksforu.blogspot.com/2013/07/c-basic-programming-c-programming.html>
- <https://tenthousandmeters.com/blog/python-behind-the-scenes-1-how-the-cpython-vm-works/>
- <https://towardsdatascience.com/how-does-python-work-6f21fd197888>
- <https://realpython.com/>