

2023

# Chuqurlashtirilgan dasturlash – PHP



Dasturchilardan  
dasturchilarga

Muallif  
**Sanjarbek Sobirjonov**

# Chuqurlashtirilgan dasturlash — php

## Mundarija

[Mundarija](#)

[Muqaddima](#)

[Muallif](#)

[Interpretatsiya va kompilyatsiyaning farqi nima?](#)

[PHP dunyosida 2022-yil oxirigacha ro'y bergan o'zgarishlar](#)

[PHP 7.\\* versiyadan PHP 8.\\* versiyaga o'tishdagi o'zgarishlar](#)

[PHP 8.\\* versiyaga qo'shilgan yangiliklar](#)

[Nomli argumentlar](#)

[Sinf o'zgaruvchilari\(xossa\)ni konstruktorda belgilash](#)

[Atributlar](#)

[Qo'shma turlar](#)

[Match operatori](#)

[Nullsafe operatori](#)

[Qo'shimcha funksiyalar va o'zgarishlar](#)

[Enum](#)

[never qaytuvchi turi](#)

[readonly o'zgaruvchilar](#)

[callable sintaksisi](#)

[Ajratalgan turlar](#)

[Ochiqlanmaydigan ma'lumotlarni yashirish](#)

[Dinamik sinf xossalari \(o'zgaruvchilar\)](#)

[Dasturlash bo'yicha nazariya va amaliyotlar](#)

[Konkurensiya, paralellik va asinxronlik](#)

[Oqim \(thread\)](#)

[Oqimlarning turlari va ularning ishlashi](#)

[Yashil oqim \(green thread\)](#)

[Stack nima?](#)

[Fiber](#)

[JIT \(Just In Time\) kompilyatsiya](#)

[Afzalliklari](#)

[Kamchiliklari](#)

[Tezlik raqamlarda](#)

FFI (Foreign Function Interface)

Yakun

## Muqaddima

Assalomu alaykum, aziz o'quvchi. Kitobimizni o'qir ekansiz, albatta tajriba olish, ilm olishni maqsad qilgansiz. Biz ham mana shu maqsadingizga yetishingizni juda ham xohlaymiz. Agar kitobdagi materiallar siz uchun foydali hisoblansa, bu biz uchun katta baxt hisoblanadi. Kitob muallif tomonidan o'z so'zi bilan yozilgan. Ya'ni, bu kitob biror chet adabiyotining tarjima varianti emas. Muallif kitobdagi har bir material o'quvchiga tushunarli bo'lishiga harakat qilgan. Kitobda atamalar **ingliz tilida keltiriladi** lekin atamalarning ma'nosi ham yo'l-yo'lakay o'zbek tilida tushuntirib ketishga harakat qilingan.

Bu kitob ko'proq ish tajribasi bor dasturchilar uchun mo'ljallangan bo'lib, tajribasini yanada oshirish maqsadida o'qishsa nur ustiga a'lo nur bo'ladi. Agar sizda hali ish tajribasi bo'lmasa bu kitob siz uchun biroz qiyinroq bo'lishi tabiiy. Boshlovchi dasturchilar uchun "**PHP7 va Obyektga yo'naltirilgan dasturlash**" nomli kitobni o'qib chiqishlarini tavsiya qilgan bo'lar edik.

Bu kitobni olgan ekansiz, uni ichida qanday ma'lumotlar borligiga qiziqishingiz tabiiy. Muallifning kitobni yozishdan maqsadi — **PHP** dasturlash tilini rivojiga hissa qo'shish hamda tildagi yangi imkoniyatlarni o'zbek tilida ham yoritishdan iborat. 2022-yilda PHP dasturlash tilida olamshumul o'zgarishlar paydo bo'ldi. Bu o'zgarishlar tufayli **PHP** endi oddiy til sifatida emas balki haqiqiy dasturlash tili sifatida ham tan olinishiga sabab bo'ladi. Keling, kitobda qanday ma'lumotlar berilganligi haqida qisqacha aytilib o'tamiz.

Kitobda **PHP kodi** qanday qilib interpretatsiya bo'lishi, **Zend** nima ekanligi, **PHP 7.4** va **PHP 8** versiyalari orasidagi tafavutlar, **PHP 8.\*** versiyalaridagi yangiliklar, **JIT (Just in Time)** nima uchun kerakligi hamda **Fiber** va **Atributlar** haqida qiziqarli ma'lumotlar keltirilgan. Bu ma'lumotlarni kitobni o'qish davomida birma-bir bilib olishingiz mumkin. Yoqimli mutolaa tilaymiz.

## Muallif

**"PHP7 va Obyektga yo'naltirilgan dasturlash", "Pythonni noldan professionalgacha o'rganing"** nomli tarjima kitoblar hamda hozirgi o'z tajribasiga tayangan holda yozilgan chuqurlashtirilgan o'zbek tilidagi kitob muallifi va **muhandis dasturchi** — **Sanjarbek Sobirjonov**. Ya'ni, o'zim 😊. Hozirda **muhandis dasturchi** bo'lib ishlayman. Biror insonga o'z bilganlarimni ularashishni juda yoqtiraman va shu boisdan kitob yozishga mehrim bor. Hozirgi o'qiyotgan kitobingizdan oldingi ikkita: **PHP** va **Python** kitoblarini tartib bilan yozish, iloji boricha o'quvchi uchun tushunarliroq

bo'lishi hamda ma'lumotlarda xatolikka yo'l qo'ymaslik uchun tarjima qilishni to'g'ri deb bilganman. Chunki men uchun asos muhim.

Hozirgi kitobda nima yozilgan bo'lsa barchasi meni shaxsiy tajribamdan kelib chiqib yozilgan. Agar bu resursdan xatolik topsangiz uni menga ulashishingizdan behad xursand bo'laman.

 **Telegram:** [@sobirjonovs](https://t.me/sobirjonovs)

 **Elektron pochta:** [sobirjonovsdev@gmail.com](mailto:sobirjonovsdev@gmail.com)

 **Github:** <https://github.com/sobirjonovs>

 **Linkedin:** <https://www.linkedin.com/in/sobirjonovsdev>

# Interpretatsiya va kompilyatsiyaning farqi nima?

Intrepretatsiya qilinadigan tillarga misol qilib **PHP**, **Python** kabilarni keltirishimiz mumkin. Bu tillarda yozilgan kodimizni ishga tushirganimizda interpretator uni har doim qaytadan o'qishiga to'g'ri keladi. Lekin kompilyatsiya qilingan tillarda, masalan **Java**, **C**, **C#** kabilarda yozilgan dasturlar bir marta kompilyatsiya bo'ladi va natijada kompilyatsiya qilingan dastur hosil bo'ladi. Bu dasturni ishga tushirganimizda bizga har doim kodimizda nima logika yozgan bo'lsak o'sha instruksiyalarning natijasini qaytaraveradi. Kodni qaytadan o'qib o'tirmaydi.

Qisqa qilib aytadigan bo'lsak, interpretator kodni natijasini qaytarsa, kompilyator kodni dastur qilib qaytaradi. Natijani ko'rish uchun esa dasturni ochish kerak bo'ladi. Kompilyator dasturni hosil qilish uchun bir marta ishga tushganligi uchun interpretatorga nisbatan ancha tez.

Interpretator bilan kompilyatorning yana bir ko'zga ko'rinishdigan farqlari, interpretator kodni **opcode(operation code, bytecode** deb ham ataladi)larga bo'lib chiqadi, kompilyator esa kodni to'g'ridan-to'g'ri assemblер tiliga yoki to'g'ridan-to'g'ri mashina tiliga o'giradi. Assemblerdagи instruksiyalarni ham **opcode** deb ataymiz deyishingiz mumkin. To'g'ri, ular ham opkod deyiladi lekin intrepretatordagi opkodlar assemblernikи bilan bir xil emas. Ular interpretator o'zi o'qiy oladigan opkodlar xolos.

**Opcode** (operatsiya kodi) - tilga xos bo'lgan amallar (operatsiyalar) nomi. Har bir amal har bir tilni dizayniga qarab har xil kod bilan nomlanadi. Masalan, **PHP** da `Z_ADD`, **python**da `BINARY_ADD`, `INPLACE_ADD` va hakozo.

Interpretatsiya qilinadigan deyarli ko'p tillarning virtual mashinasi bo'ladi. Masalan, **The Python Virtual Machine**, **Zend Engine Virtual Machine** va boshqalar. Bu virtual mashinalarning vazifasi opkodlarni mashina tiliga o'girib chiqishdan iborat. Opkodlar qayerdan hosil bo'lar edi? Albatta, interpretatordan. Interpretator bizning kodimizni o'qiydi, uni opkodlarga bo'lib chiqadi va virtual mashinaga uzatadi. Virtual mashina kerakli buyruqlarni bajarib bizga natijani qaytarib beradi. Ko'p interpretatorlarning hayot sikli shundan iborat.

Har bir interpretatsiya qilingan tillardagi virtual mashinalarning opkodlari bir-biridan farq qiladi. Masalan assembler, python virtual mashina va php virtual mashinasi tushunadigan opkodlarga diqqat bilan e'tibor qarating.

Assembly Language	This is the same as machine language, except the command numbers have been replaced by letter sequences which are more readable and easier to memorize.	<p><b>AT&amp;T:</b></p> <pre>push %ebp sub \$0x8,%esp movb \$0x41,0xffffffff(%ebp)</pre> <p><b>Intel:</b></p> <pre>push ebp mov ebp, esp sub esp, 0C0h</pre> <p><b>HLA (High Level Assembly):</b></p> <pre>program HelloWorld; #include( "stdlib.hhf" ) begin HelloWorld; stdout.put( "Hello, World of Assembly Language", nl ); end HelloWorld;</pre>
-------------------	---	--

### Assembler opkodlari

<https://www.tenouk.com/Bufferoverflowc/Bufferoverflow1b.html>

57h	POP_BLOCK	0	Removes one block from the block stack. Per frame, there is a stack of blocks, denoting nested loops, try statements, and such.	for a in (1,2): break  for a in (1,2): a = 1 a = 2 finally: a = 3  try: a = 1 except ValueError: a = 2 finally: a = 3  class a: pass  <code object>  a = 1  del a  (a, b) = "ab"
				 00000000 78 (0F 00) - SETUP_LOOP 00000003 64 (03 00) = 00000044 TUPLE: (00000049 INT: ...) - LOAD_CONST 00000006 44 - GET_ITER 00000007 50 - POP_TOP 00000008 5A (00 00) - 00000058 STR: 'a' (01 00 00 00 61) - STORE_NAME 0000000D 50 - BREAK_LOOP 0000000E 71 (0F 00) - POP_BLOCK_ABSOLUTE 00000011 57 - POP_BLOCK 00000012 64 (02 00) = 00000043 None (48) - LOAD_CONST 00000013 53 - RETURN_VALUE  00000000 7A (2A 00) - SETUP_FINALLY 00000003 79 (09 00) - SETUP_EXCEPT 00000006 64 (00 00) = 00000059 INT: 1 (01 00 00 00) - LOAD_CONST 00000007 5A (00 00) - 00000070 STR: 'a' (01 00 00 00 61) - STORE_NAME 00000008 57 - POP_BLOCK 00000009 6E (19 00) - JUMP_FORWARD 0000000A 64 (00 00) = 00000056 None (48) - LOAD_CONST 0000000B 65 (01 00) = 00000076 STR: 'ValueError' (0A 00 00 00 56 61 6C 75 65 45 72 72...) - LOAD_NAME 0000000C 6A (09 00) = 'EXC_MATCH' - COMPARE_OP 0000000D 67 (0F 00) - POP_JUMP_IF_FALSE 00000011 01 - POP_TOP 00000012 64 (01 00) = 00000040 INT: 2 (02 00 00 00) - LOAD_CONST 00000013 5A (00 00) = 00000070 STR: 'a' (01 00 00 00 61) - STORE_NAME 00000014 66 (02 00) - JUMP_FORWARD 00000018 01 - POP_TOP 00000019 64 (01 00) = 00000046 INT: 3 (03 00 00 00) - LOAD_CONST 00000020 5A (00 00) = 00000070 STR: 'a' (01 00 00 00 61) - STORE_NAME 00000021 59 - BUILD_LIST 00000022 64 (02 00) = 00000065 None (48) - LOAD_CONST 00000024 66 (02 00) - JUMP_FORWARD 00000028 58 - END_FINALLY 00000029 57 - POP_BLOCK 0000002D 64 (01 00) = 00000065 None (48) - LOAD_CONST 0000002D 64 (03 00) = 00000066 INT: 3 (03 00 00 00) - LOAD_CONST 00000031 5A (00 00) = 00000070 STR: 'a' (01 00 00 00 61) - STORE_NAME 00000032 59 - BUILD_CLASS 00000033 64 (02 00) = 00000065 None (48) - LOAD_CONST 00000037 53 - RETURN_VALUE  00000000 65 (00 00) = 00000088 STR: 'name' (08 00 00 00 5F 5F 61 6D 45 5F 5F) - LOAD_NAME 00000003 5A (01 00) = 00000075 STR: '__module' (0A 00 00 00 5F 5F 6D 6F 64 75 6C 65...) - STORE_NAME 00000007 53 - LOAD_LOCALS 00000009 53 - RETURN_VALUE  <code object>  00000000 64 (00 00) = 0000002D INT: 1 (01 00 00 00) - LOAD_CONST 00000003 5A (00 00) = 00000038 STR: 'a' (01 00 00 00 61) - STORE_NAME 00000004 64 (01 00) = 00000032 None (48) - LOAD_CONST 00000009 53 - RETURN_VALUE  00000000 64 (00 00) = 00000030 INT: 1 (01 00 00 00) - LOAD_CONST 00000003 5A (00 00) = 00000038 STR: 'a' (01 00 00 00 61) - STORE_NAME 00000004 64 (01 00) = 0000003B STR: 'a' (01 00 00 00 61) - DELETE_NAME 00000005 64 (01 00) = 00000035 None (48) - LOAD_CONST 0000000C 53 - RETURN_VALUE  00000000 64 (00 00) = 00000033 STR: 'ab' (02 00 00 00 61 ...) - LOAD_CONST 00000003 5C (02 00) - UNPACK_SEQUENCE 00000004 64 (01 00) = 00000040 STR: 'a' (01 00 00 00 61) - STORE_NAME 00000005 5A (01 00) = 00000041 STR: 'b' (01 00 00 00 62) - STORE_NAME 00000009 64 (01 00) = 0000003A None (48) - LOAD_CONST 0000000D 53 - RETURN_VALUE  00000000 78 (0F 00) - SETUP_LOOP 00000003 64 (01 00) = 00000043 TUPLE: (00000048 INT: ...) - LOAD_CONST 00000006 44 - GET_ITER
				TOS is an iterator. Call its next() method. If this yields a

## Python opkodlari

<https://unpyc.sourceforge.net/Opcodes.html>

Number	Name	Has sample code?
0	<u>NOP</u>	yes
1	<u>ADD</u>	yes
2	<u>SUB</u>	yes
3	<u>MUL</u>	yes
4	<u>DIV</u>	yes
5	<u>MOD</u>	yes
6	<u>SL</u>	yes
7	<u>SR</u>	yes
8	<u>CONCAT</u>	yes
9	<u>BW_OR</u>	yes
10	<u>BW_AND</u>	yes
11	<u>BW_XOR</u>	yes
12	<u>BW_NOT</u>	yes
13	<u>BOOL_NOT</u>	yes
14	<u>BOOL_XOR</u>	yes
15	<u>IS_IDENTICAL</u>	yes
16	<u>IS_NOT_IDENTICAL</u>	yes
17	<u>IS_EQUAL</u>	yes
18	<u>IS_NOT_EQUAL</u>	yes
19	<u>IS_SMALLER</u>	yes
20	<u>IS_SMALLER_OR_EQUAL</u>	yes
21	<u>CAST</u>	yes
22	<u>QM_ASSIGN</u>	yes
23	<u>ASSIGN_ADD</u>	yes
24	<u>ASSIGN_SUB</u>	yes
25	<u>ASSIGN_MUL</u>	yes
26	<u>ASSIGN_DIV</u>	yes
27	<u>ASSIGN_MOD</u>	yes
28	<u>ASSIGN_SL</u>	yes
29	<u>ASSIGN_SR</u>	yes

PHP opkodlari

<http://php.adamharvey.name/manual/en/internals2.opcodes.php>

## PHP dunyosida 2022-yil oxirigacha ro'y bergan o'zgarishlar

Xabaringiz bor, **Rasmus Lerdorf** PHP dasturlash tilining asoschisi hisoblanadi. **PHP** ning boshlanishini Rasmus og'amiz yozib bergani bilan keyingi versiyalari boshqa

dasturchilar tomonidan davom ettirilmoqda. Ko'pchilik **PHP** dasturlash tilini yomon ko'rishadi. Negaligi menga qorong'u. Lekin shuni aytish joizki, yomon til yo'q, yomon dasturchi bor. Tilni vosita sifatida ko'rish — bu eng to'g'ri yo'l. Kerakli muammoga kerakli qurolni tanlay olgan dasturchi har doim yutadi.

**2020-yil 26-noyabrd** hamma kutayotgan **PHP 8** dunyoga keldi. Bu versiya juda katta xursandchilik bilan qarshi olindi. Chunki unda hamma kutayotgan o'zgarishlar mavjud edi. Lekin uni ortidan yomon xabar bizni kutayotgan edi...

**PHP** tiliga eng ko'p hissa qo'shayotgan asosiy dasturchi **Nikita Popov** **PHP** tilidagi o'z faoliyatini yakunlashga qaror qildi. Bu ish **2021-yil dekabr** oylarida ro'y berdi. Shundan so'ng ko'pgina PHP muxlislari bu xabarni qayg'u bilan qarshi oldi. Bunday katta yo'qotish **PHP**ni xafa qilishi tabiiy albatta. Lekin **PHP** muxlislari chekinishni istamadi.

**Jetbrainsda** faoliyat olib boruvchi dasturchi, **Roman Pronskiy** "The PHP Foundation" fondini tuzdi. Bu fonddan maqsad PHP dasturlash tilini rivojlanishini to'xtab qolmasligi hamda uni kengroq tadbiq etishdan iborat edi va bu sekin-sekin o'z natijasini bera boshladgi. Fondga katta-katta kompaniyalar, dasturchilar va oddiy insonlar donat qila boshlashdi. Yig'ilgan donatlar esa PHP tilining asosiy dasturchilariga maosh sifatida berilmoqda.

**PHP** jamoasida yarim stavka, to'liq stavkada ishlaydigan dasturchilar bor. Ular o'z mehnatlari uchun pul olganlari bilan ularning mehnati faqat o'zlarinigina qornini to'ydirmaydi, balki barcha **PHP** ishtiyoqmandlarining ham qora qozonini qaynatadi. Ularga bu mehnatlari uchun tashakkur deymiz. Agar siz ham fondga o'z hissangizni qo'shamoqchi bo'lsangiz marhamat: <https://thephp.foundation>

Fond tuzilganidan so'ng **PHP** tilida juda ko'p yangiliklar chiqa boshladgi. Yangi-yangi qo'shimchalar hamda tuzatishlar **PHP** muxlislарini xursand qila boshladgi. Kitob davomida o'sha o'zgarishlarni sizlar bilan ko'rib chiqamiz.

## **PHP 7.\* versiyadan PHP 8.\* versiyaga o'tishdagi o'zgarishlar**

Dasturlash tillarida "deprecations" va "removals" degan tushuncha bor. Ya'ni kelajakda olib tashlanishi mumkin bo'lgan funksiyalar **deprecated** deb qo'yiladi. Olib tashlanganlari esa **removed**. Agar biror tilni dokumentatsiyasida bu funksiya eskirgan ya'ni deprecated degan yozuvni ko'rsangiz, bu funksiyani ishlatishga shoshilmay turing yoki alternativini topib ishlatiting.

PHP 8.\* versiya chiqqanidan so'ng PHP 7.\* versiyadagi ko'p narsalari o'zgardi. Masalan quyidagi eskirgan funksiyalarga qarang:

## Deprecated Features

### PHP Core

- If a parameter with a default value is followed by a required parameter, the default value has no effect. This is deprecated as of PHP 8.0.0 and can generally be resolved by dropping the default value, without a change in functionality.

```
<?php
function test($a = [], $b) {} // Before
function test($a, $b) {}     // After
?>
```

One exception to this rule are parameters of the form `Type $param = null`, where the null default makes the type implicitly nullable. This usage remains allowed, but it is recommended to use an explicit nullable type instead:

```
<?php
function test(A $a = null, $b) {} // Still allowed
function test(?A $a, $b) {}      // Recommended
?>
```

1. Agar funksiyamizda majburiy parametrlardan oldin boshlang'ich qiymatga ega bo'lган ixtiyoriy parametrlar mavjud bo'lsa unda quyidagicha ogohlantirish qaytadi:

The screenshot shows the PHP Sandbox interface. In the code editor, there is a warning message: "Deprecated: Optional parameter \$a declared before required parameter \$b is implicitly treated as a required parameter in /home/user/scripts/code.php on line 3". Below the code editor, there is a "Result for 8.2.1" section which displays the same warning message. At the bottom right of the result area, it says "Execution time: 0.000088s Mem: 388KB Max: 427KB".

Lekin bu ogohlantirish xatolik emas va dastur ish faoliyatini to'xtatmaydi. Lekin buni hozirdan to'g'rilamasangiz kelgusida albatta to'xtatishi aniq. Buni to'g'rilash uchun `$a`

parametrimizdagi boshlang'ich qiymatni olib tashlashimiz yoki `null` ma'lumot turiga o'zgartirishimiz kerak.

### Standard Library

- Sort comparison functions that return `true` or `false` will now throw a deprecation warning, and should be replaced with an implementation that returns an integer less than, equal to, or greater than zero.

```
<?php
// Replace
usort($array, fn($a, $b) => $a > $b);
// With
usort($array, fn($a, $b) => $a <= $b);
?>
```

2. `usort` funksiyasi massivlarni foydalanuvchi belgilagan shart asosida tartiblash uchun ishlatiladi. Oldingi versiyalarda funksiyadan mantiqiy qiymat (`true/false`) qaytishi kerak edi. Lekin **PHP 8.0** dan boshlab bu funksiya foydalanuvchining funksiyasidan kichkina, katta yoki tenglikni ifodalovchi butun sonni qaytishini kutadi. Bu qanaqa bo'ladi deysizmi? Hozir!

The screenshot shows a PHP Sandbox interface. The code area contains:

```
1 <?php
2 $a = 5;
3 $b = 6;
4
5 # Kichkina
6 var_dump($a <= $b);
7
8 # Katta
9 var_dump($b <= $a);
10
11 # Teng
12 var_dump($b <= $b);
```

The result area shows the output of the code execution:

```
int(-1)
int(1)
int(0)
```

Execution time: 0.000102s Mem: 387KB Max: 425KB

PHPda `"<=>"` kosmik kema nomli operator mavjud. Bu operator xuddi shu natijani qaytarish uchun ishlatiladi. Agar birinchi argument ikkinchi argumentdan kichkina bo'lsa `-1`, agar ikkinchi argument birinchi argumentdan katta bo'lsa `1`, agar birinchi va ikkinchi argument teng bo'lsa `0` butun sonlarini qaytaradi. `usort` funksiyasi ham endi xuddi shunday sonlarni kutadi.

3. Agar siz phpda **Reflection API** bilan ishlagan bo'lsangiz demak bu narsani tushunasiz. Mayli, shunda ham qisqacha ma'lumot berib ketaman. **Reflection API** bilan siz "magic" ya'ni sehrli ish ishlar qila olasiz. Qanaqa deysizmi? Masalan, **PHP** funksiyasini parametrlarini o'qiy olishingiz, interfeyslarda nima metodlar borligini bilishingiz, qaysi sinf qaysi sinfdan ko'chirma olyotganini bilishingiz mumkin. Va undan tashqari Laravel freymworkingin asosiy qurollaridan bo'lgan **Dependency Injection** xususiyatini ham shu **Reflection API** bilan birga o'zingiz o'z qo'lingiz bilan amalga oshirolasiz. Laravelni ichida inyeksiya qilish uchun **Reflection API** hamda **DI** konteyner ishlataladi. Xo'p, mavzudan chetlashmaylik.

#### Reflection

- [ReflectionFunction::isDisabled\(\)](#) is deprecated, as it is no longer possible to create a [ReflectionFunction](#) for a disabled function. This method now always returns **false**.
- [ReflectionParameter::getClass\(\)](#), [ReflectionParameter::isArray\(\)](#), and [ReflectionParameter::isCallable\(\)](#) are deprecated. [ReflectionParameter::getType\(\)](#) and the [ReflectionType](#) APIs should be used instead.

Funksiya parametrlarini turini aniqlash uchun oldin `:: isArray` yoki `:: isCallable` metodlari ishlatalardi. Endi bular ham eskiribdi. Uni o'rniغا `:: getType` metodini ishlatasiz.

**Eskirishlar (deprecations)** — dastur ish faoliyatini buzmaydi. Lekin yangi versiyaga o'tmoqchi bo'lsangiz, eskirgan funksionalni yangisiga moslashtirmaslik keyinchalik bu eskirishlar olib tashlangandan so'ng yaxshi oqibatga olib kelmaydi.

Eskirishlarga misollar shulardan iborat. Bu faqatgina misol emas balki real misol hisoblanadi. **PHP 7.\*** versiyadan **PHP 8.\*** versiyaga o'tishda shularni inobatga olib qo'yishingiz kerak.

E'tiborga olishga arziydi. **PHPda** qaysi xatolik kodi dastur ish faoliyatini buzadi, qaysini buzmasligini bilib oling

Value	Constant	Description	Note
1	<code>E_ERROR</code> (int)	Fatal run-time errors. These indicate errors that can not be recovered from, such as a memory allocation problem. Execution of the script is halted.	
2	<code>E_WARNING</code> (int)	Run-time warnings (non-fatal errors). Execution of the script is not halted.	
4	<code>E_PARSE</code> (int)	Compile-time parse errors. Parse errors should only be generated by the parser.	
8	<code>E_NOTICE</code> (int)	Run-time notices. Indicate that the script encountered something that could indicate an error, but could also happen in the normal course of running a script.	
16	<code>E_CORE_ERROR</code> (int)	Fatal errors that occur during PHP's initial startup. This is like an <code>E_ERROR</code> , except it is generated by the core of PHP.	
32	<code>E_CORE_WARNING</code> (int)	Warnings (non-fatal errors) that occur during PHP's initial startup. This is like an <code>E_WARNING</code> , except it is generated by the core of PHP.	
64	<code>E_COMPILE_ERROR</code> (int)	Fatal compile-time errors. This is like an <code>E_ERROR</code> , except it is generated by the Zend Scripting Engine.	
128	<code>E_COMPILE_WARNING</code> (int)	Compile-time warnings (non-fatal errors). This is like an <code>E_WARNING</code> , except it is generated by the Zend Scripting Engine.	
256	<code>E_USER_ERROR</code> (int)	User-generated error message. This is like an <code>E_ERROR</code> , except it is generated in PHP code by using the PHP function <a href="#">trigger_error()</a> .	
512	<code>E_USER_WARNING</code> (int)	User-generated warning message. This is like an <code>E_WARNING</code> , except it is generated in PHP code by using the PHP function <a href="#">trigger_error()</a> .	
1024	<code>E_USER_NOTICE</code> (int)	User-generated notice message. This is like an <code>E_NOTICE</code> , except it is generated in PHP code by using the PHP function <a href="#">trigger_error()</a> .	
2048	<code>E_STRICT</code> (int)	Enable to have PHP suggest changes to your code which will ensure the best interoperability and forward compatibility of your code.	
4096	<code>E_RECOVERABLE_ERROR</code> (int)	Catchable fatal error. It indicates that a probably dangerous error occurred, but did not leave the Engine in an unstable state. If the error is not caught by a user defined handle (see also <a href="#">set_error_handler()</a> ), the application aborts as it was an <code>E_ERROR</code> .	
8192	<code>E_DEPRECATED</code> (int)	Run-time notices. Enable this to receive warnings about code that will not work in future versions.	
16384	<code>E_USER_DEPRECATED</code> (int)	User-generated warning message. This is like an <code>E_DEPRECATED</code> , except it is generated in PHP code by using the PHP function <a href="#">trigger_error()</a> .	
32767	<code>E_ALL</code> (int)	All errors, warnings, and notices.	

Yaxshi, eskirgan funksional yozilgan bo'lsa ham dasturimiz buzilmas ekan. Lekin yana bitta shunday narsa bor-ki, u eski kodlarni ishlamay qolishiga sabab bo'ladi. Bu narsa, **"backward incompatible changes"**, ya'ni **"oldingilari bilan mos bo'limgan o'zgarishlar (qisqartirilgan shakli — ombo')"** deyiladi. Istalgan tilning yangi versiyasi chiqar ekan, yo eskirgan funksional yo ombo', yoki ikkalasi ham bo'lishi aniq. **PHP** dasturlash tilining yangi versiyasida ham ba'zi ombo'lar mavjud. Bularni birma-bir ko'rib chiqamiz. Kettik.

1. Ko'pchilik php dasturchilari bilishlari kerak. `0 == "foo"` qanday natija qaytaradi?

Albatta hozir `true` deyishingiz mumkin va bu to'g'ri. Lekin aslida bu narsa noto'g'ri.

Qanday qilib 0 fooga teng bo'lishi mumkin? Bu narsa yangi versiyada to'g'rildi va endi oldindi yozgan kodlaringizda mana shunday taqqoslash qilgan bo'lsangiz

noto'g'ri natija olishi hamda mavjud kodingizni ish faoliyatini boshqacha ishlashiga sabab bo'ladi. O'zgarishlarni rasmida ko'rishingiz mumkin:

#### String to Number Comparison

Non-strict comparisons between numbers and non-numeric strings now work by casting the number to string and comparing the strings. Comparisons between numbers and numeric strings continue to work as before. Notably, this means that `0 == "not-a-number"` is considered false now.

Comparison	Before	After
<code>0 == "0"</code>	<code>true</code>	<code>true</code>
<code>0 == "0.0"</code>	<code>true</code>	<code>true</code>
<code>0 == "foo"</code>	<code>true</code>	<code>false</code>
<code>0 == ""</code>	<code>true</code>	<code>false</code>
<code>42 == "42"</code>	<code>true</code>	<code>true</code>
<code>42 == "42foo"</code>	<code>true</code>	<code>false</code>

- Agar siz obyektlaringizda ma'lum o'zgaruvchilarni borligini tekshirmoqchi bo'lsangiz, `isset` yoki `property_exists` funksiyalarini ishlatishingiz kerak. Oldin obyekt o'zgaruvchisi borligini tekshirish uchun `array_key_exists` funksiyasini ishlatgan bo'lsangiz endi bu kodingiz ishlamaydi.

- The ability to use `array_key_exists()` with objects has been removed. `isset()` or `property_exists()` may be used instead.

- Endi statik bo'limgan metodlaringizni statik metod kabi chaqirolmaysiz. Bu imkoniyat yangi versiyada ham olib tashlandi. Agar statik bo'limgan metodlaringizni statik chaqirayotgan bo'lsangiz darhol o'zgartiring.

- The ability to call non-static methods statically has been removed. Thus `is_callable()` will fail when checking for a non-static method with a classname (must check with an object instance).

- 8-chi** versiyagacha bo'lgan barcha kodlarda, bilsangiz, sinfni ichida sinf nomi bilan bir xil bo'lgan metod yozsangiz, php uni konstruktor deb tushunardi. Endi bu eskilik sarqti ham olib tashlandi. Endi u oddiy metod deb o'qiladi:

- Methods with the same name as the class are no longer interpreted as constructors. The `__construct()` method should be used instead.

5. `match` va `mixed` so'zlari endi tilning zaxiradagi maxsus instruksiyasi hisoblanadi.

Endi bu nom bilan funksiya yoki sinf yaratgan bo'lsangiz, uni nomini o'zgartirishingizga to'g'ri keladi aks holda dasturingiz fatal xatolik otadi:

- `match` is now a reserved keyword.
- `mixed` is now a reserved word, so it cannot be used to name a class, interface or trait, and is also prohibited from being used in namespaces.

6. `(real)` va `(unset)` kabi kastlarni ishlatgan bo'lsangiz, bular ham yangi versiyada olib tashlangan.

- The `(real)` and `(unset)` casts have been removed.

7. Agar siz o'zingizni shaxsiy sinf yuklovchi (**автозагрузка классов**) `__autoload` funksiyasi bilan amalga oshirgan bo'lsangiz, endi bu funksiya yangi versiyada olib tashlangan. O'rniqa `spl_autoload_register` funksiyasi ishlatalishi shart:

- The ability to specify an autoloader using an `__autoload()` function has been removed. `spl_autoload_register()` should be used instead.

8. Eski php versiyalarida oldin ma'lumotlarni iteratsiya qilish uchun `each` funksiyasi ishlatalardi. Endi bu funksiya ham yangi versiyadan olib tashlandi:

- `each()` has been removed. `foreach` or `ArrayIterator` should be used instead.

9. PHP muhit o'zgaruvchilari faylidagi `track_errors` direktivasi fayldan olib tashlandi.

Endi `$php_errormsg` o'zgaruvchisini ishlatib bo'lmaydi. Uni o'rniqa `error_get_last()` funksiyasini ishlatalamiz ekan:

- The `track_errors` ini directive has been removed. This means that `php_errormsg` is no longer available. The `error_get_last()` function may be used instead.

10. Konstantalarni `define` funksiyasi bilan belgilashda, uni katta yo kichikligini tekshiruvchi parametrini o'chirib qo'ysa bo'lar edi. Bu bu parametr ham olib tashlandi:

- The ability to define case-insensitive constants has been removed. The third argument to `define()` may no longer be `true`.

11. `error_reporting` funksiyasini boshlang'ich qiymati endi `E_ALL`. Oldin esa `E_DEPRECATED` va `E_NOTICE` bo'lgan edi:

- The default `error_reporting` level is now `E_ALL`. Previously it excluded `E_NOTICE` and `E_DEPRECATED`.

12. Bu operatorni yaxshi bilsangiz kerak. Xato chiqishini xohlamay qolgan paytimizda ishlatardik yoshligimizda 😊. @ operatorini ishlatganimizda xatolik chiqmas edi va `error_reporting` chaqirilganida bu funksiyadan 0 qiymat qaytar edi. Endi yangi versiyada bu bekor qilindi. Agar siz ham shunday tekshiruv qilgan bo'lsangiz endi buni o'zgartirishingizga to'g'ri keladi:

- The @ operator will no longer silence fatal errors (`E_ERROR`, `E_CORE_ERROR`, `E_COMPILE_ERROR`, `E_USER_ERROR`, `E_RECOVERABLE_ERROR`, `E_PARSE`). Error handlers that expect `error_reporting` to be 0 when @ is used, should be adjusted to use a mask check instead:

```
<?php
// Replace
function my_error_handler($err_no, $err_msg, $filename, $linenum) {
    if (error_reporting() == 0) {
        return false;
    }
    // ...
}

// With
function my_error_handler($err_no, $err_msg, $filename, $linenum) {
    if (!(error_reporting() & $err_no)) {
        return false;
    }
    // ...
}
?>
```

13. Endi #[ bilan boshlanuvchi izohlar izoh sifatida emas balki atribut sintaksisi sifatida interpretatsiya qilinadi. **Atributlar** yangi versiyada qo'shilgan qo'shimcha. Kitobni o'rtalarida bu haqida gaplashamiz:

- `#[` is no longer interpreted as the start of a comment, as this syntax is now used for attributes.

14. Bir sinfdan ko'chirma qilayotganda asosiy sinfdagi metod nomini ikkinchi sinfda boshqacha tur bilan yozganda eski versiyada ogohlantirish chiqarar edi. Endi yangi versiyada bu narsa **fatal** xatolik chiqaradi. Chunki bu “ogohlantirish” **LSP (Liskov almashtirish tamoyili)**ni buzyapti. Mana nega phpni sevaman. Dizayn andozalariga mos kelishga iloji boricha harakat qiladi:

```

1  <?php
2
3  class Otaxon {
4      /**
5      * @param int $haqida
6      * @return void
7      */
8      public function gapirmoqda(int $haqida): void
9      {
10
11     }
12 }
13
14 class Bolajon extends Otaxon {
15     /**
16     * @param string $haqida
17     * @return void
18     */
19     public function gapirmoqda(string $haqida): void
20     {
21
22     }
23 }
24
25 echo 'Lekin kodimiz ishlayveradi!';

```

Oldingi natija:

```

sobirjonovs@MacBook-Pro-Sanjarbek test % php book.php
PHP Warning: Declaration of Bolajon::gapirmoqda(string $haqida): void should be compatible with Otaxon::gapirmoqda(int $haqida): void in /Users/sobirjonovs/Projects/test/book.php on line 19
Warning: Declaration of Bolajon::gapirmoqda(string $haqida): void should be compatible with Otaxon::gapirmoqda(int $haqida): void in /Users/sobirjonovs/Projects/test/book.php on line 19
Lekin kodimiz ishlayveradi!

```

Hozir esa:

```

PHP Fatal error: Declaration of Bolajon::gapirmoqda(string $haqida): void must be compatible with Otaxon::gapirmoqda(int $haqida): void in /Users/sobirjonovs/Projects/test/book.php on line 19
Fatal error: Declaration of Bolajon::gapirmoqda(string $haqida): void must be compatible with Otaxon::gapirmoqda(int $haqida): void in /Users/sobirjonovs/Projects/test/book.php on line 19

```

E'tibor bergen bo'lsangiz, "**Lekin kodimiz ishlayveradi!**" matni yangi versiyada chiqmayapti. Chunki bu xatolik butun skript ish faoliyatini to'xtatadi.

15. Qo'shish, ayirish va bitlarni siljitim amallarida birlashtirish operatorining ustunligi o'zgargan. Ya'ni endi oshiqcha qavslar yozish shart emas:

- The precedence of the concatenation operator has changed relative to bitshifts and addition as well as subtraction.

```
<?php
echo "Sum: " . $a + $b;
// was previously interpreted as:
echo ("Sum: " . $a) + $b;
// is now interpreted as:
echo "Sum:" . ($a + $b);
?>
```

Aytaylik `$a` o'zgaruvchining qiymati 4 va `$b` o'zgaruvchiniki 5. Birinchi bajarilayotgan amalda, `"Sum: " . $a + $b` yozilgan. Eski versiyada bu amalning natijasi 5 chiqadi. Nega? Chunki `. + -` operatorlarining ustunligi yangi versiyada bir xil edi. Shuning uchun bu amallar ketma-ket bajarildi. `"Sum: " . $a` bu birinchi bajariladi. Natijada `"Sum: 4"` hosil bo'ladi. Ikkinci amal esa `"Sum: 4" + $b` shunday ko'rinishga keladi. Eski versiyada satrni songa qo'shsa satr Oga aylanib qolishini oldin aytib o'tgan edim. Shunday ekan, bu amalda `0 + 5` hosil bo'ladi bu esa 5 ga aylanadi. Tushundingiz-a? Endi yangi versiyada esa bu narsa to'g'rilangan. Birinchi bo'lib qo'shish amali keyin esa birlashtirish bajariladigan bo'ldi. Operatorlar ustunligi ro'yxati:

Operator Precedence		
Associativity	Operators	Additional Information
(n/a)	clone new	<a href="#">clone</a> and <a href="#">new</a>
right	**	<a href="#">arithmetic</a>
(n/a)	+ - ++ -- ~ (int) (float) (string) (array) (object) (bool) @	<a href="#">arithmetic (unary + and -)</a> , <a href="#">increment/decrement</a> , <a href="#">bitwise</a> , <a href="#">type casting</a> and <a href="#">error control</a>
left	instanceof	<a href="#">type</a>
(n/a)	!	<a href="#">logical</a>
left	* / %	<a href="#">arithmetic</a>
left	+ - .	<a href="#">arithmetic (binary + and -)</a> , <a href="#">array</a> and <a href="#">string</a> (. prior to PHP 8.0.0)
left	<< >>	<a href="#">bitwise</a>
left	.	<a href="#">string</a> (as of PHP 8.0.0)
non- associative	< <= > >=	<a href="#">comparison</a>
non- associative	== != === !== <> <=>	<a href="#">comparison</a>
left	&	<a href="#">bitwise</a> and <a href="#">references</a>
left	^	<a href="#">bitwise</a>
left		<a href="#">bitwise</a>
left	&&	<a href="#">logical</a>
left		<a href="#">logical</a>
right	??	<a href="#">null coalescing</a>
non- associative	? :	<a href="#">ternary</a> (left-associative prior to PHP 8.0.0)
right	= += -= *= **= /= .= %= &=  = ^= <<= >>= ??=	<a href="#">assignment</a>
(n/a)	yield from	<a href="#">yield from</a>
(n/a)	yield	<a href="#">yield</a>
(n/a)	print	<a href="#">print</a>
left	and	<a href="#">logical</a>
left	xor	<a href="#">logical</a>
left	or	<a href="#">logical</a>

Ro'yxatdagi tartiblar ustunlikni bildiradi. Tepa o'rinda turgan operator pastdag'i operatorlardan ustunligini bildiradi. **"Associativity"** esa bo'g'anishni bildiradi. Ya'ni, chap tomondagi qiymatga bog'lanamidimi yoki o'ng, shu narsani ifodalaydi. **"Operators"** esa o'sha o'rinda turgan narsa qaysi operatorlar ekanligini ifodalaydi.

Eski versiyadan yangi versiyaga o'tishda ko'proq muhim deb bilgan narsalarimni yozishga harakat qildim. Agar yanada ko'proq ma'lumot olmoqchi bo'lsangiz PHP saytiga kirib, rus yoki ingliz tilida o'qib chiqishingiz mumkin.

# PHP 8.\* versiyaga qo'shilgan yangiliklar

O'zim python dasturlash tilida ham ishlaganligim uchun, python dasturlash tilidagi ayrim imkoniyatlarni php dasturlash tilida bo'lishini ham xohlar edim.

## Nomli argumentlar

U imkoniyatlardan biri — bu **nomli argumentlar** edi. Nomli argumentlar nima? Biz eski versiyada funksiyadagi parametrlarni nomi bilan argument berishimizning iloji yo'q edi.

```
<?php

function nomli_argument(string $ism, string $familiya): void
{
    echo $ism . ' ' . $familiya;
}

// PHP 8.* versiyadan oldin
nomli_argument(ism: 'Sanjarbek', familiya: 'Sobirjonov');

// PHP 8.* versiyalarda
nomli_argument(ism: 'Sanjarbek', familiya: 'Sobirjonov');
```

Oldin funksiyada ixtiyoriy parametrlar bo'lsa, ularni ham yozib o'tirishimiz kerak edi. Kerakli argumentlarni berolmasdik. Endi esa bemalol faqatgina kerakli parametrlarga argument bera oladigan bo'ldik.

## Sinf o'zgaruvchilari(xossa)ni konstruktorda belgilash

Oldinlari bu narsa ko'p vaqtimizni o'g'rilib qo'yar edi. Konstruktorda yozilgan parametrlarni qiymatini sinfning o'zgaruvchisiga belgilash kerak edi. Yangi versiyada bunday qilish shart emas. Yangi versiyada konstruktorni o'zida sinf o'zgaruvchisini yozib, bira to'la uning ko'rinish sohasini ham belgilab ketish mumkin.

**Ko'rinish sohasi (visibility, modicator)** — bu xossaning sinf ichida yoki tashqarisida ishlatalish chegaralarini belgilaydi. Xossa, ya'ni sinf o'zgaruvchisi agar **ochiq (public)** bo'lsa, unda u o'zgaruvchi tashqarida ham ishlatalishi mumkin. Agar

**himoyalangan (protected)** bo'lsa, tashqarida ishlatib bo'lmaydi lekin sinfdan ko'chirma olayotgan pastki sinflar ichida ishlatsa bo'ladi. Agar **yopiq (private)** bo'lsa, faqatgina o'sha sinfning o'zidagina ishlatsa bo'ladi.

```
/**  
 * @mixin UserService  
 */  
trait ShowInformation {  
    /**  
     * @return string  
     */  
    public function show(): string  
    {  
        $result = 'Ism: ' . $this->name;  
        $result .= PHP_EOL;  
        $result .= 'Viloyat: ' . $this->region;  
        $result .= PHP_EOL;  
        $result .= 'Yosh: ' . $this->age;  
  
        return $result;  
    }  
}  
  
// PHP 8.* versiyadan oldin  
class UserServiceOld {  
    use ShowInformation;  
  
    public string $name;  
    public string $region;  
    public int $age;  
  
    public function __construct(string $name, string $region, int $age)  
    {  
        $this->name = $name;  
        $this->region = $region;  
        $this->age = $age;  
    }  
}  
  
// PHP 8.* versiyalarda  
class UserService {  
    use ShowInformation;  
  
    public function __construct(  
        public string $name,  
        public string $region,  
        public int $age  
    ) {}  
}  
  
$oldService = new UserServiceOld( name: 'Sanjarbek Sobirjonov', region: 'Sirdaryo', age: 20);  
$service = new UserService(name: 'Sanjarbek Sobirjonov', region: 'Sirdaryo', age: 20);
```

Ha aytgancha, yangi versiyada nomli argumentlarni faqatgina funksiyalarda emas balki metodlarda ham ishlatsa bo'ladi.

## Atributlar

Yangi versiyaga qo'shilgan yana bir ajoyib imkoniyatlardan biri — bu atributlar hisoblanadi. Bu ham python dasturlash tilida yoki boshqa dasturlash tillarida bor bo'lgan kerakli xususiyatlardan biri. Masalan, python dasturlash tilida bu narsa **dekorator** deb ataladi. Agar python kitobimni sotib olgan bo'lsangiz yoki python dasturchi bo'lsangiz bu narsani albatta qanday ishlashini bilasiz. PHP da ham bu deyarli shunday ishlaydi. Endi bilmaganlar bo'lsa birga o'rghanamiz.

PHP hamjamiyatida tilga yangi imkoniyat qo'shish kerak bo'lsa rasmiy saytida so'rovnoma o'tkazsa bo'ladi. Bu so'rovnoma **RFC (Request for Comment)** deyiladi. Agar so'rovnomada aytilgan xususiyat ko'pchilik tarafidan ma'qullansa, bu imkoniyat phpga qo'shilishi mumkin.



## PHP RFC: Attributes v2

- Version: 0.5
- Date: 2020-03-09
- Author: Benjamin Eberlei ([beberlei@php.net](mailto:beberlei@php.net)), Martin Schröder
- Status: Implemented
- Target: 8.0
- First Published at: [http://wiki.php.net/rfc/attributes\\_v2](http://wiki.php.net/rfc/attributes_v2)
- Implementation: <https://github.com/php/php-src/pull/5394>

Large credit for this RFC goes to Dmitry Stogov whose previous work on attributes is the foundation for this RFC and patch.

### Introduction

This RFC proposes Attributes as a form of structured, syntactic metadata to declarations of classes, properties, functions, methods, parameters and constants. Attributes allow to define configuration directives directly embedded with the declaration of that code.

Similar concepts exist in other languages named **Annotations** in Java, **Attributes** in C#, C++, Rust, Hack and **Decorators** in Python, Javascript.

So far PHP only offers an unstructured form of such metadata: doc-comments. But doc-comments are just strings and to keep some structured information, the @-based pseudo-language was invented inside them by various PHP sub-communities.

On top of userland use-cases there are many use-cases for attributes in the engine and extensions that could affect compilation, diagnostics, code-generation, runtime behavior and more. Examples are given below.

The wide spread use of userland doc-comment parsing shows that this is a highly demanded feature by the community.

### Proposal

#### Attribute Syntax

Attributes are a specially formatted text enclosed with "<<" and ">>" by reusing the existing tokens T\_SL and T\_SR.

attributes may be applied to many things in the language:

Ko'rib turganingizdek, tilga qo'shilayotgan har bir imkoniyat albatta so'rovnomanidan o'tkaziladi. Atributlar ham shunday so'rovnomanadan muvaffaqiyatli o'tti. Bu so'rovnomada atribut qo'shilishi tarafdarlari soni jami 92ta bo'lди, xohlamaganlar esa 13ta bo'lди. Lekin atributlarni belgilashdagi sintaksisida juda ko'p muammolar paydo bo'lди. Sintaksisi uchun ham alohida so'rovnoma o'tkazildi, naqd alohida 2ta so'rovnoma o'tkazildi. Ko'pchilik `@@()` sintaksisi tarafdarlari bo'lsa, ba'zilari `@()` sintaksisi tarafdarlari bo'lди. Lekin sintaksis shunday bo'lganida u juda ko'p muammolar tug'dirishi mumkin edi. Qanday muammo? Siz bilan @ operatori haqida gaplashgandik. Bu operator xato chiqsa uni ko'rsatmaslikka javobgar edi. Eski versiyalarda shunday kodlar bor-ki, u yerda funksiyadan qaytayotgan natija uchun shu operator ishlatilgan:

```

<?php

/**
 * @returnmixed
 */
function oldingi_versiyada() {
    throw new RuntimeException('Dastur ish faoliyatida xatolik');
}

@@oldingi_versiyada(); // yoki
@oldingi_versiyada();

```

Ko'rdingizmi? Agar atribut sintaksisini shunday bo'lganida, juda ko'p muammolar tug'dirgan bo'lar edi. Shuning uchun ular `#[]` sintaksisini ma'qullahdan boshqa chora topa olmadilar. Ha aytgancha, atributlarni nomlashda ham ancha muhokama bo'lgan 😊 Ba'zilar atribut nomini ma'qullagan bo'lsa, ba'zilar annotatsiya nomini ma'qullagan. Oxiri, atribut nomi tarafdorlari yutib chiqdi.

Mavzuga qaytaylik o'rtoqlar. Atribut nima o'zi? Atribut bu o'zgaruvchiga, metodga, sinfga va boshqalarga uni o'zgartirmasdan qo'shimcha funksional qo'shuvchi vosita hisoblanadi. Atributlarni belgilash uchun yuqoridaqidek sintaksisni kerakli manzilning tepasiga yozib qo'yish kifoya. Biror sinfni atribut qilish uchun birinchi o'sha sinfga atribut yoziladi.

```

<?php

#[Attribute]
class MeningAtributim {
    // bu yerda o'ta kuchli logikali
    // kodlar yozilgan deb tasavvur qiling
}

```

Mana endi `MeningAtributim` atributimizni istalgan joyga atribut sintaksisi orqali yozib qo'yaverishimiz mumkin:

```

<?php

#[Attribute(Attribute::TARGET_FUNCTION)]
class MeningAtributim {
    // bu yerda o'ta kuchli logikali
    // kodlar yozilgan deb tasavvur qiling
}

```

```

#[MeningAtributim]
function atributlangan() {
    //
}

```

Bo'ldi, atributni ishlattik. Endi bu atributlangan funksiyamizni ishlatish uchun atributimizga kerakli logika yozishimiz kerak. U logikada funksiya qanday holatda chaqirilishi, nimalar bajarilishi yoziladi. Shundan so'ng, atributimizdan obyekt yaratib, o'sha logika yozilgan metodimizni chaqirib qo'yamiz. Ha, to'g'ri. Funksiyani o'zini chaqirganingizda bu narsa ishlamaydi. Albatta, atributni o'zini ishlatish kerak:

```

<?php

#[Attribute(Attribute::TARGET_FUNCTION)]
class MyAttribute {
    /**
     * @param array $data
     */
    public function __construct(public array $data = []) {}

    /**
     * @return void
     */
    public function superLogic(): void
    {
        $reflectionFunction = new ReflectionFunction('attributed');

        foreach ($reflectionFunction->getAttributes() as $attribute) {
            // It is only for target checking
            $newAttribute = $attribute->newInstance();

            echo json_encode($newAttribute->data) . PHP_EOL;
            $reflectionFunction->invoke();
        }
    }
}

#[MyAttribute(['attributed function attribute'])]
function attributed(): void
{
    echo 'original content of this function';
}

(new MyAttribute)->superLogic();

```

Natija esa:

```
>>> ["attributed function attribute"]
>>> original content of this function
```

Atributlarimizni qayerda ishlatischimiz uchun cheklov qo'yishimiz ham mumkin. Bu cheklovlarni atributimizni belgilayotgan payt, **Attribute** sinfiga kerakli argumentlarni berish orqali amalga oshiramiz.

```
<?php

#[Attribute(Attribute::TARGET_FUNCTION, Attribute::IS_REPEATABLE)]
class MyAttribute {
    /**
     * @param array $data
     */
    public function __construct(public array $data = []) {}

    /**
     * @return void
     */
    public function superLogic(): void
    {
        $reflectionFunction = new ReflectionFunction('attributed');

        foreach ($reflectionFunction->getAttributes() as $attribute) {
            // It is only for target checking
            $newAttribute = $attribute->newInstance();

            echo json_encode($newAttribute->data) . PHP_EOL;
            $reflectionFunction->invoke();
        }
    }
}

// Agar atributni takror ishlatmoqchi bo'lsangiz, IS_REPEATABLE
// argumentni tepada yozib qo'yishingiz kerak
#[MyAttribute(['attributed function attribute'])]
#[MyAttribute(['attributed function attribute'])]
function attributed(): void
{
    echo 'original content of this function';
}

(new MyAttribute)->superLogic();
```

Atributlarga beriladigan argumentlar — nishonlar deb ataladi. Atributlarning 6ta nishoni mavjud:

- **Attribute::TARGET\_CLASS** - Faqat sınıf uchun
- **Attribute::TARGET\_FUNCTION** - Faqat funksiya uchun
- **Attribute::TARGET\_METHOD** - Faqat metod uchun
- **Attribute::TARGET\_PROPERTY** - Faqat xossa uchun
- **Attribute::TARGET\_CLASS\_CONSTANT** - Faqat sınıf konstantasi uchun
- **Attribute::TARGET\_PARAMETER** - Faqat parameter uchun (metod yo funksiyaning)
- **Attribute::TARGET\_ALL** - Agar hech qaysisi belgilanmasa, boshlang'ich qiymati shunga teng bo'ladi. Ya'ni, hammasi uchun ishlayveradi.

Atributlarni ishlatish bo'yicha hayotiy misolga yaqin bo'lgan namunani **Github** repozitorimga kirib ko'rishingiz mumkin: <https://github.com/sobirjonovs/php-attributes-sample>

## Qo'shma turlar

Eski versiyalarda eng kerak bo'layotgan xususiyatlardan yana biri — bu birlashma turlar (union types) edi. Baxtga ko'ra, bu xususiyat ham yangi PHP 8.0 versiyaga qo'shildi. Endi biz bir nechta turga oid ma'lumotni birgina o'zgaruvchida qabul qilishimiz mumkin!

```
<?php

function unionTypes(int|float $age) {
    echo 'Yosh: ' . $age;
}

unionTypes(20);
echo PHP_EOL;
unionTypes(19.5);
```

Bir nechta turlarni qo'shish uchun `|` operatori ishlatiladi. Agar siz turlarni qo'lda yozib o'tirishni xohlamasangiz, istalgan turni qabul qilmoqchi bo'lsangiz unda mixed turini ishlatishingiz mumkin. Bu ham yangi versiyada paydo bo'lgan xususiyat:

```
<?php

function unionTypes(mixed $age) {
    echo 'Yosh: ' . $age;
}
```

```
unionTypes(20);
echo PHP_EOL;
unionTypes(19.5);
```

Xo'p, buni bilib oldik. Endi murakkabroq variantini ko'rib chiqamiz. Bu variant algebradagi **dizunksiya** va **konyunksiya** mavzulari bilan bog'liq. Algebradan agar oz bo'lса ham xabaringiz bo'lса buni tushunishingiz qiyin bo'lmaydi.

**Matematika** — butun sohalarning tayanchi va shu jumladan dasturlashni ham. Matematikani dasturlashdan ayirgan dasturchi, yaxshi dasturchi emas.

### Dizunksiya va konyunksiya nima?

Agar ikkita mulohaza "yoki" so'zi bilan bog'lansa, unda mulohazalar dizunksiyasi hosil bo'ladi. Masalan, **p** va **q** mulohazalari **dizunksiyasi** **p v q** kabi belgilanadi.

Masalan:

**p:** **Samsung** kompaniyasining eng yaxshi telefon modeli Iphone 14 Pro Max Ultra

**q:** **Apple** kompaniyasining eng yaxshi telefon modeli Iphone 14 Pro Max Ultra

**p V q:** **Samsung** kompaniyasining eng yaxshi telefon modeli Iphone 14 Pro Max Ultra **YOKI** **Apple** kompaniyasining eng yaxshi telefon modeli Iphone 14 Pro Max Ultra.

Dizunksiyada mulohazalardan biri haqiqat bo'ladigan bo'lса, bu mulohazamiz haqiqat bo'ladi.

Agar ikkita mulohaza "va" so'zi bilan bog'lansa, unda mulohazalar konyunksiyasi hosil bo'ladi. Masalan, **p** va **q** mulohazalari **konyunksiyasi** **p ^ q** kabi belgilanadi.

Masalan:

**p:** **Samsung** kompaniyasining eng yaxshi telefon modeli Iphone 14 Pro Max Ultra

**q:** **Apple** kompaniyasining eng yaxshi telefon modeli Iphone 14 Pro Max Ultra

**p ^ q:** **Samsung** kompaniyasining eng yaxshi telefon modeli Iphone 14 Pro Max Ultra **VA** **Apple** kompaniyasining eng yaxshi telefon modeli Iphone 14 Pro Max Ultra.

Konyunksiyada mulohazalardan biri yolg'on bo'ladigan bo'lса, bu mulohazamiz yolg'on bo'ladi.

Biz bularni dasturlashda odatda har kuni ishlatalamiz. Biz ishlataladigan `||` va `&&` operatorlari shunday vazifalarni bajaradi.

Xo'p, murakkabroq variantni ko'rib chiqmoqchi edik. "Chalg'ib" qoldik. Birlashgan turlarni ko'rib chiqdik, endi biz kesishgan turlarni ko'rib chiqamiz. Kesishma turlarni **& (va)** bitli operatori bilan ifodalaymiz. Bu operatorimiz ham konyunksiyaga misol bo'la oladi. Kesishgan tur bilan birlashgan turni nima farqi bor deyishingiz mumkin, albatta. Farqi bitta, birlashgan turlar **| (or)** bitli operatori bilan ifodalansa, kesishgan turlar **& (va)** bitli operatori bilan ifodalanadi. Birlashgan turda, argument yoki bu yoki u turga mansub bo'lishi kerak, kesishgan turda argument esa ikkala turning ham funksionallariga ega bo'lishi kerak. Buni chuqurroq misoldak o'rib o'tamiz:

```
<?php

interface Frontend {
    /**
     * @return string
     */
    public function html(): string;

    /**
     * @return string
     */
    public function javascript(): string;

    /**
     * @return string
     */
    public function vue(): string;
}

interface Backend {
    /**
     * @return string
     */
    public function php(): string;

    /**
     * @return string
     */
    public function postgres(): string;

    /**
     * @return string
     */
    public function redis(): string;
}
```

```

interface UserInterface {
    /**
     * @return string
     */
    public function display():string;
}

class MonolithApp implements Frontend, Backend, UserInterface {

    /**
     * @return string
     */
    public function html(): string
    {
        return 'html';
    }

    /**
     * @return string
     */
    public function javascript(): string
    {
        return 'javascript';
    }

    /**
     * @return string
     */
    public function vue(): string
    {
        return 'vue';
    }

    public function php(): string
    {
        return 'php';
    }

    public function postgres(): string
    {
        return 'postgres';
    }

    public function redis(): string
    {
        return 'redis';
    }

    /**
     * @return string
     */
    public function display(): string
    {
        $data = $this->html() . PHP_EOL;

```

```

        $data .= $this->javascript() .PHP_EOL;
        $data .= $this->vue() .PHP_EOL;
        $data .= $this->php() .PHP_EOL;
        $data .= $this->postgres() .PHP_EOL;
        $data .= $this->redis() .PHP_EOL;

        return $data;
    }
}

/**
 * @param Backend&UserInterface&Frontend$app
 * @return void
 */
function createApp(Frontend & Backend & UserInterface $app): void
{
    echo $app->display();
}

createApp(new MonolithApp());

```

`createApp()` funksiyamizda `Frontend & Backend & UserInterface` interfeyslarni realizatsiya qilgan sinfni kutyotgan parametrni ko'rishimiz mumkin. Ko'rib turganingizdek, `MonolithApp` degan sinfimiz mana shu uchala interfeys bilan ishlashga majbur va ularsiz ishi bitmaydi. `createApp` funksiyamizga ham mana shunday monolit dastur yaratadigan sinf kerakligini belgilab qo'ydik. Mana shu tur kesishgan tur deyiladi. Bu turda uchala turdag'i metodlar ham kelayotgan argumentda bor bo'lishi shart. Aks holda, fatal xatolik qaytaradi.

Hozir siz bilan ko'rib chiqqan turimiz **PHP 8.1** versiyaga qo'shilgan. Endi keling, biroz murakkablashtirilgan variantini ham ko'ramiz. Bu variant "**qo'shma turlar**" va "**kesishma turlar**" birlashmasi desak mubolag'a bo'lmaydi. Uning nomi **DNF (eng. Disjunctive normal forms, o'zb. Dizunktiv normal shakllar)** turi deb ataladi.

Dizunktiv normal shakllar [https://en.wikipedia.org/wiki/Disjunctive\\_normal\\_form](https://en.wikipedia.org/wiki/Disjunctive_normal_form) shu yerda qanday formulada ko'rsatilgan bo'lsa, PHP dasturlash tilidagi turda ham xuddi shunday ishlaydi. Bu mavzu matematikaning elementar bo'limlarining biri hisoblanadi.

Xo'p, agar hali o'qimagan bo'lsangiz, shu yerda ham tushuntirib o'tib ketaman. Dizunktiv normal shakllarni qisqa konyunksiyalar dizunksiyasi deyishimiz mumkin. U qanday bo'ladi?

- (A va B) yoki (C va D)

- A yoki B
- A yoki B yoki (D va C)

Demak, dizyunktiv normal shakllarning hammasi mana shunga o'xshash ko'rinishda bo'ladi. PHP dasturlash tilida ham xuddi shunday bo'ladi. Bu formulani buzsangiz PHP dasturida **parsing** xatolik kelib chiqadi. Kodga qarang:

```
<?php

interface Application {

}

interface Frontend {

    public function html(): string;

    public function javascript(): string;

    public function vue(): string;
}

interface Backend {

    public function php(): string;

    public function postgres(): string;

    public function redis(): string;
}

interface UserInterface {

    public function display():string;
}

class MonolithApp implements Backend, UserInterface, Application {

    public function html(): string
    {
        return 'html';
    }

    public function javascript(): string
    {
        return 'javascript';
    }

    public function vue(): string
    {
```

```

        return 'vue';
    }

    public function php(): string
    {
        return 'php';
    }

    public function postgres(): string
    {
        return 'postgres';
    }

    public function redis(): string
    {
        return 'redis';
    }

    public function display(): string
    {
        $data = $this->html() .PHP_EOL;
        $data .= $this->javascript() .PHP_EOL;
        $data .= $this->vue() .PHP_EOL;
        $data .= $this->php() .PHP_EOL;
        $data .= $this->postgres() .PHP_EOL;
        $data .= $this->redis() .PHP_EOL;

        return $data;
    }
}

function createApp(Application $app): (Frontend&Backend)|(Frontend&UserInterface)|(Backend
&UserInterface)
{
    return $app;
}

var_dump(createApp(new MonolithApp()));

```

Oldingi kodimga biroz o'zgartirish kirittim. U yerga qo'shimcha `Application` nomli interfeys yarattim. biz endi `createApp` funksiyamizni yanada optimalroq qildik. Oldin `createApp` nomli funksiyamiz faqatgina monolit dastur yaratish uchun ishlatilayotgan edi, funksiyani nomi esa umumiy turgan edi ya'ni u funksiya ham monolit ham monolit bo'limgan dastur qaytarishi kerak edi va hozir bu funksiyamiz endi o'z vazifasini a'llo darajada bajaryapti. Funksiyamizga `Application` interfeysini realizatsiya qilayotgan obyektni yuboramiz va u funksiya obyekt ustida kerakli amallarni bajaradi va bizga dasturni ishga tushirib qaytaradi. Funksiyamizdan bizga qaytishi mumkin bo'lgan dastur

turlarini qat'iy belgilab oldik. U turlar: “**Frontend va Backend**” yoki “**Frontend va UserInterface**” yoki “**Backend va UserInterface**” dan iborat. Funksiyamizga dasturni yuborar ekanmiz, u dasturimiz yuqoridagi turlardan biriga mansub bo'lishi shart aks holda fatal xatolik chiqadi.

Bir komponentdan kamroq cheklovga ega komponent qo'shiladigan bo'lsa php fatal xatolik beradi:

```
function createApp(Application $app): (Frontend&Backend)|(Frontend&UserInterface)|(Backend&UserInterface)|Backend
{
    return $app;
}
```

Agar `Backend` interfeysimizni o'zini yozib qo'yadigan bo'lsak, yuqoridagi gapimizga ko'ra xatolik otishi kerak bo'ladi. Buni sinab ko'rishingiz ham mumkin. Chunki `Backend` interfeysimiz Frontend bilan birga shart bilan bog'langan, shu sababli `Backend` interfeysimiz yana bir shart bilan qayta bog'lanishi mumkin lekin endi hech qachon bir o'zi bo'lolmaydi:

```
function createApp(Application $app): (Frontend&Backend)|(Frontend&UserInterface)|(Backend&UserInterface)|(Backend&Application)
{
    return $app;
}
```

Mana endi bu ishlaydi.

## Match operatori

**PHP 8.0** versiyada yangi `match` operatori paydo bo'ldi. Bu operator `switch` operatori bilan deyarli bir xil. Bu operator switchdan farqli o'laroq qiymatlarni `==` ga emas, balki `==` ga tekshiradi hamda qiymat qaytaradi.

```
<?php

$number = '1';

switch ($number) {
    case 1:
        echo 'Qiymati birga teng';
```

```

        break;
    case '1':
        echo 'Qiymati birga teng';
        break;
    default:
        echo 'Defolt';
    }

// Natija: "Qiymati birga teng"

echo match ($number) {
    1 => 'Tur va qiymat bir',
    '1' => 'Tur boshqa, qiymat bir',
    default => 'Default'
};

// Natija: "Tur boshqa, qiymat bir"

```

Ko'rib turganingizdek, `match` operatorida ham default nomli buyrug'i bor. Bu buyruq xuddi `switch` dagidek bir xil vazifani bajaradi.

## Nullsafe operatori

PHP 8.0 versiyada hamma kutayotgan yana bir xususiyat paydo bo'ldi. Bu nullsafe operatori hisoblanadi. Bu operatorni javascriptchilar yaxshi biladi. Bu operatorni vazifasi null bo'lgan obyektni tekshirishdan iborat. Bu operatorni funksional ko'rinishi `is_null()` hisoblanadi. Yaxshisi namunani bir ko'rsangiz, yaxshiroq tushunasiz:

```

<?php

// manba: php.net

// nullsafe operatori bilan
$result = $repository?->getUser(5)?->name;

// is_null() funksiyasi bilan
if (is_null($repository)) {
    $result = null;
} else {
    $user = $repository->getUser(5);
    if (is_null($user)) {
        $result = null;
    } else {
        $result = $user->name;
    }
}

```

## Qo'shimcha funksiyalar va o'zgarishlar

- Endi obyektda ham `::class` operatorini ishlatalish mumkin:

```
<?php

class Payme {
    public const MERCHANT_ID = '*45kjsandlkbn2324jkklasjn';
    public static string $key = 'klansdfkldansfjnkj345345';
}

$payme = new Payme;

echo $payme::class;

// Natija: Payme
```

- `new` va `instanceof` bilan endi ifodalarni ham ishlatsa bo'ladi!!! Maza!

```
<?php

class Payme {
    public const MERCHANT_ID = '*45kjsandlkbn2324jkklasjn';
    public static string $key = 'klansdfkldansfjnkj345345';
}

$payme = new ((function () {
    return 'Payme';
})());

echo $payme::class;
```

Ajoyib-a?)

- Bungachi, nima deysiz?

```
<?php

class Payme {
    public const MERCHANT_ID = '*45kjsandlkbn2324jkklasjn';
    public static string $key = 'klansdfkldansfjnkj345345';
}

class Payment {
    public const GATEWAY= 'Payme';
}
```

```
echo Payment::GATEWAY::$key.PHP_EOL;
echo Payment::GATEWAY::MERCHANT_ID;
```

Nima bo'layotganini tushunmay qoldingizmi? 😊 Xavotir olmang. Endi konstantadagi sinfning konstantasiga yoki statik o'zgaruvchisiga yana yoki statik metodlariga bemalol shu tarzda kirsak bo'ladi. `GATEWAY` konstantamizda Payme sinfining nomi yozilgan, `GATEWAY` ga kirgandan so'ng, Payme sinfining `$key` yoki `MERCHANT_ID` konstantalarini ishlataladi.

4. Agar siz sinfingizda `__toString` metodini yozgan bo'lsangiz, endi u sinf avtomatik tarzda `Stringable` interfeysi realizatsiya qiladi. Buni tekshirish uchun shu kodni ishlatib ko'ring:

```
<?php

class Payme {
    public const MERCHANT_ID = '*45kjsandlkbn2324jkklasjn';
    public static string$key= 'klansdfkldansfjnkj345345';

    public function __toString(): string
    {
        return self::$key;
    }
}

var_dump(new Payme instanceof Stringable); // true
```

5. `Trait` ichida endi yopiq abstrakt metodlarni yozishimiz mumkin.

```
<?php

trait PaymeMixin {
    abstract private function pay();
}

class Payme {
    use PaymeMixin;

    public const MERCHANT_ID = '*45kjsandlkbn2324jkklasjn';
    public static string$key= 'klansdfkldansfjnkj345345';

    public function __toString(): string
    {
        return self::$key;
```

```
}

    public function pay(): void {}
}
```

## 6. 3tasi birda!

- throw can now be used as an expression. That allows usages like:

```
<?php
$fn = fn() => throw new Exception('Exception in arrow function');
$user = $session->user ?? throw new Exception('Must have user');
```

- An optional trailing comma is now allowed in parameter lists.

```
<?php
function functionWithLongSignature(
    Type1 $parameter1,
    Type2 $parameter2, // <-- This comma is now allowed.
) {
}
```

- It is now possible to write catch (Exception) to catch an exception without storing it in a variable.

Endi `throw` ham ifoda sifatida ishlatalishi mumkin ekan. Undan tashqari, funksiyamizni oxirgi parametrini vergul bilan tugatish imkon qo'shilibdi. Yana bir menga yoqqan xususiyatlardan biri bu `catch` qismida istisnoni o'zgaruvchiga olmasdan turib tutib olish:

```
<?php

trait PaymeMixin {
    abstract private function pay();
}

class Payme {
    use PaymeMixin;

    public const MERCHANT_ID = '*45kjsandlkbn2324jkklasjn';
    public static string$key= 'klansdfkldansfjnkj345345';

    public function __toString(): string
    {
        return self::$key;
    }

    public function pay(): void {}
}
```

```
try {
    $payme = new Payme;
} catch (Exception) {
    error_log('payme exception', 3, 'error.log');
}
```

7. 3 xil kerakli funksiyalar ham qo'shilgan: `str_contains`, `str_starts_with` `str_ends_with`.

`str_contains` — 1-parametrga qayerdan qidirilishi, 2-parametrga nimani qidirish kerakligini kiritiladi. Agar 2-argument 1-argument ichidan topilsa, `true`, aks holda `false` qaytaradi.

`str_starts_with` — Buniki ham `str_contains` bilan bir xil. Faqat, agar 1-argumentning boshlanishi 2-argument bilan bir xil bo'lsa, true, aks holda false qaytaradi.

`str_ends_with` — Bu ham `str_starts_with` bilan deyarli bir xil, faqat oxirini tekshiradi.

## Enum

**Enum**lar PHP 8.1 dan boshlab paydo bo'ldi. **Enum** yoki **Enumeratsiyalar** deb bir turga oid bo'lган qiymatlar ro'yxati hisoblanadi. Enumeratsiyalar bo'lmasidan oldin biz bunday xususiyatlarni sinfga konstanta qo'shish orqali ishlatishga odatlangan edik:

```
<?php

interface PaymentInterface {
    public const PAID = 1;
    public const UNPAID = 2;
}
```

Endi esa bular eskirdi. Enumeratsiyalar chiqqanidan so'ng, bular yanada ixchamlashdi. PHP da enumeratsiyalar ikki xil: turlangan va turlanmagan enumeratsiyalardan iborat. Turlanmagan enumeratsiyalar o'zi nomi bilan turlanmagandir. Uning element qiymatlari bo'lmaydi faqat elementlardan tashkil topgan bo'ladi. Enumning elementlari `case` kalit so'zi bilan belgilanadi. Agar `case` kalit so'zi bilan belgilansa hamda unga qiymat belgilansa, u turlangan elementga aylanadi, shu bilan birga enumning o'zi ham turlangan enumga aylanib qoladi. Turlangan enumning ichida faqatgina turlangan elementlar bo'lishi mumkin. Xuddi shunday, turlanmagan enumda ham faqatgina turlanmagan elementlar bo'lishi mumkin. Turlangan enumlar faqatgina `int` yoki `string` ma'lumot turi bilan ifodalanishi mumkin:

```
<?php

enum PaymentMethods: int
{
    case PAYME= 1;
    case CLICK= 2;
    case UZUM= 3;
    case BANK= 4;
}
```

Turlanmagan enumlarda shunchaki qiymat bo'lmaydi, xolos:

```
<?php

enum PaymentMethods
{
    case PAYME;
    case CLICK;
    case UZUM;
    case BANK;
}
```

Faqat shu bilan cheklanib qolamizmi? Qo'shimcha metod yoki konstantalar qo'sholmaymizmi degan savol kelayotgan bo'lsa, kuttirmayman. Bemalol, qo'shsangiz bo'ladi:

```
<?php

enum PaymentMethods
{
    case PAYME;
    case CLICK;
    case UZUM;
    case BANK;

    public const PAYME_CONST = self::PAYME;
    public const PAYME_MERCHANT_ID = '324234234';

    public function get(): string
    {
        return match ($this) {
            self::PAYME=> 'Payme',
            self::CLICK=> throw new Exception('To be implemented'),
            self::UZUM=> throw new Exception('To be implemented'),
            self::BANK=> throw new Exception('To be implemented')
        };
    }
}
```

```
    }  
}
```

Enumning har elementini konstanta kabi ishlatsa (**masalan: PaymentMethods::PAYME**) bo'ladi. Enum elementi o'zi ham enum. Ya'ni, har bir element o'sha enumning obyekti hisoblanadi. Enumning o'zini kerakli metod va xossalari mavjud:

```
<?php  
  
enum PaymentMethods  
{  
    case PAYME;  
    case CLICK;  
    case UZUM;  
    case BANK;  
  
    public const PAYME_CONST = self::PAYME;  
    public const PAYME_MERCHANT_ID = '324234234';  
  
    public function get(): string  
    {  
        return match ($this) {  
            self::PAYME=> 'Payme',  
            self::CLICK=> throw new Exception('To be implemented'),  
            self::UZUM=> throw new Exception('To be implemented'),  
            self::BANK=> throw new Exception('To be implemented')  
        };  
    }  
}  
  
var_dump(PaymentMethods::cases());  
var_dump(PaymentMethods::PAYME->name);  
  
// Agar turlangan enum bo'lsa, valuesini shu bilan olsa bo'ladi.  
var_dump(PaymentMethods::PAYME->value);
```

## Enumlarda nimalar bo'lmaydi?

1. **Konstruktur va destruktur**
2. Ko'chirma olish va berish (**extend**)
3. Statik va statik bo'lмаган о'згарувчilar (xossa)
4. Belgilangan enumdan oldin uni ishlatish

## Nimalar bo'ladi?

1. `public`, `protected` va `private` metodlar
2. `public`, `protected` va `private` konstantalar
3. `public`, `protected` va `private` statik metodlar
4. Interfeysi realizatsiya qilish
5. `__call`, `__callStatic`, and `__invoke` sehrli metodlari

## never qaytuvchi turi

`never` qaytuvchi turi ham **PHP 8.1** dan boshlab paydo bo'ldi. Agar bilsangiz `void` qaytuvchi turi ham bor edi. U ham funksiyadan qiymat qaytmasligi uchun belgilanadigan tur hisoblanadi. Lekin `never` nega kerak?

`never` turi funksiyadan umuman ma'lumot qaytmasligi uchun kerak. Agar funksiyangizda siz return ishlatmasdan shunchaki echo qilib yozib qo'yadigan bo'lsangiz yoki boshqa logikalar yozsangiz ham, funksiyangizdan `NULL` qaytib keladi. Bu esa funksiyangiz bevosita ma'lumot qaytarayotganini isbotlaydi. Shuning uchun never turi qo'shilgan. never turi ishlataligan funksiya yo `die`, `exit` yoki `throw` bilan istisno otishi kerak bo'ladi. Uni ishlatishdan maqsad undan keyingi kodlarni ishlashiga xalaqit berishdan iborat. Men sizlarga kichkinagina, realroq misol tayyorlab qo'ydim:

```
<?php

class Computer {
    public int $battery = 10;
    public bool $working = false;

    public string $game;

    public function powerOn(): static
    {
        $this->working = true;

        echo "Kompyuter yondi";

        return $this;
    }

    public function powerOff(): never
    {
        $this->working = false;

        die("Kompyuter o'chirildi");
    }
}
```

```

    public function playGame(string $name): static
    {
        $this->game = $name;

        echo "O'yin boshlandi";

        return $this;
    }
}

$computer = new Computer();

$computer->powerOn();

/**
 * Kompyuter o'chsa, hech qanday amal bajarilmasin.
 */
$computer->powerOff();
$computer->playGame('NFS');

```

Bu tur ham **RFC** dan o'tgan. Ko'pchilik buni nomlanishini `noreturn` bo'lishligini xohlagan lekin oxirida `never` nomi g'olib kelgan.

`never` qaytuvchi turi **birlashgan** turlarda yoki parametr turlarida ishlatilmaydi. Aks holda, fatal xatolik yuzaga keladi.

## readonly o'zgaruvchilar

**php 8.1** versiyada `readonly` degan yangi kalit so'z qo'shildi. Bu kalit so'zning vazifasi sinf o'zgaruvchilarini faqat o'qish uchun belgilab qo'yishdan iborat. `readonly` qilingan o'zgaruvchilarni o'zgartirish, unga yozish mumkin emas aks holda fatal xatolik yuzaga keladi. `readonly` o'zgaruvchilarni faqatgina bir marta konstruktorda qiymat belgilash mumkin, undan keyin uni o'zgartirishni iloji yo'q:

```

<?php

class My Readonly {
    public readonly string $name;

    public function __construct(string $name)
    {
        $this->name = $name;
    }
}

$readonly = new My Readonly("Sanjarbek Sobirjonov");

```

```
echo $readonly->name;

$readonly->name = 'Hello'; // PHP Fatal error: Uncaught Error: Cannot modify readonly property My Readonly::$name in
```

**php 8.2** versiyada esa sinflarni o'zini ham `readonly` qilsa bo'ladi. Sinf `readonly` qilinganidan so'ng, uni ichidagi o'zgaruvchilar ham avtomatik `readonly` o'zgaruvchilar bo'lib qoladi:

```
<?php

readonly class MyReadonly {
    public string $name;

    public function __construct(string $name)
    {
        $this->name = $name;
    }
}

$readonly = new MyReadonly("Sanjarbek Sobirjonov");
echo $readonly->name;

$readonly->name = 'Hello'; // PHP Fatal error: Uncaught Error: Cannot modify readonly property My Readonly::$name in
```

## callable sintaksi

Agar esingizda bo'lsa, biz funksiyalardan **callable** yaratish uchun

`Closure::fromCallable('funksiyanomi')` qilib ishlatar edik. Endi esa bunday qilib o'tirishimiz shart emas. php 8.1 versiyadan boshlab callable yaratish sintaksisi qo'shildi. Endilikda **callableni** `funksiyanomi(...)` qilib yaratsak bo'ladi:

```
<?php

$print_r = print_r(...);

$print_r(['name' => 'Sanjarbek Sobirjonov']);
```

Buni **nullsafe** operatorlarda yoki **obyekt** yaratayotganda ishlatib bo'lmaydi.

## Ajratilgan turlar

Ajratilgan turlar deyishimga sabab bor. **PHP 8.2** versiyada endi bool turini emas, balki true va false, null kabi turlarni ham ishlatsa bo'ladi ekan:

```
function returnFalse():false {}

function returnNull():null {}

function returnTrue():true {}
```

## Ochiqlanmaydigan ma'lumotlarni yashirish

PHP 8.2 versiyada endi debug qilayotganda parol, token va kalit kabi ko'rsatilishi kerak bo'limgan ma'lumotlarni yashirib qo'yishingiz mumkin. Buni `#[\SensitiveParameter]` atributi yordamida amalga oshirasiz:

```
function generatePassword(#[\SensitiveParameter]string $password)  {
    debug_print_backtrace();
}

generatePassword('hunter2');

/**
array(1) {
[0]=> array(4) {
    ["file"]=> string(38) "..."
    ["line"]=> int(9)
    ["function"]=> string(3) "generatePassword"
    ["args"]=> array(1) {
        [0]=> object(SensitiveParameterValue)#1 (0) {}
    }
}
}
```

Ko'rib turganingizdek, funksiya argumentini qiymatini ko'rsatmayapti. Agar oddiy string deb yozib qo'ysangiz, sizning parolingizni ham ko'rsatib qo'ygan bo'lar edi.

## Dinamik sinf xossalari (o'zgaruvchilar)

**PHP 8.2** versiyada dinamik belgilanadigan o'zgaruvchilar endi eskirgan funksionallar sirasiga kiritildi. Agar mabodo siz xossalarni \_\_get yoki \_\_set magic metodlari orqali dinamik belgilamoqchi bo'lsangiz unda bu atributni `#[\AllowDynamicProperties]` ishlatishtingiz kerak. Aks holda, sizga error chiqaradi.

```
# [AllowDynamicProperties]
class User {
    private int $id;
}

$user = new User();
$user->name = 'Sanjar';

/**
Deprecated: Creation of dynamic property User::$name is deprecated in ... on line ...
**/
```

Endi sal jiddiyroq mavzularga o'tsaylik-a?

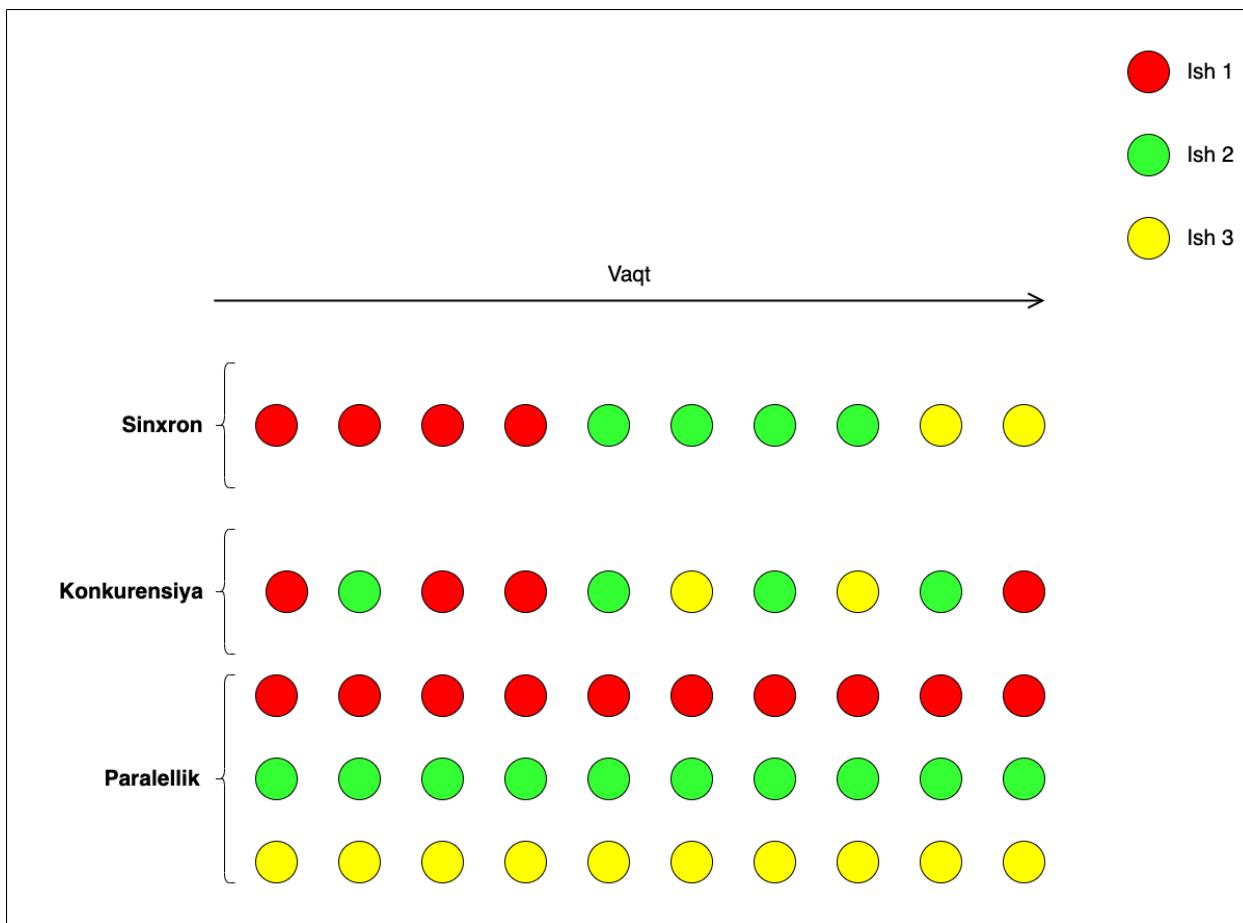
## Dasturlash bo'yicha nazariya va amaliyotlar

### Konkurensiya, paralellik va asinxronlik

Bu mavzuyimiz keyingi mavzularga bog'liq. Keyingi mavzularni tushunish uchun shular haqida qisqacha ma'lumot bo'lishi kerak. Xo'p, konkurensiya, paralellik va asinxron kabi so'zlarni tanishlaringizdan eshitgan bo'lsangiz kerak. Eng kamida, asinxron so'zini aniq eshitgansiz. **"E PHP da asinxron yo'q"**, **"PHP bo'lmaydi"**, **"Faqat sinxron ishlaydi"** shunga o'xshash malomat qilgan gaplarini ham eshitgandirsiz. PHP-da nega asinxronlik hali ham yo'q degan savolga javobni kitob davomida sekin-sekin yoritib boraman.

1. **Konkurensiya** — bir nechta ishni bir vaqt ichida bajarilishiga konkurensiya deyiladi.
2. **Sinxron** — bir nechta ishni ketma-ket, bir-birini kutgan holda bajarilishiga sinxron deyiladi.
3. **Paralellik** — bir nechta ishni alohida, bir vaqt ni o'zida birga bajarilishiga paralellik deyiladi.

Tushunarliroq bo'lishi uchun diagramaga qarang. Rang bilan belgilagan doiralarim bular biror dasturiy vazifa hisoblanadi:



Muallif - o'zim

**Konkurensiya** va **paralellik** keng tushunchalar bo'lib, ular o'z ichiga biz bilgan tushunchalarni ham oladi. Xo'p, **konkurensiyaga** nima kiradi? Asinxron ishlaydigan kodni konkurent deb aytishimiz ham mumkin. Chunki asinxron kodimizni **eventloopga** qo'yib, eng tez bajarilgan ishdan rezultat olib, keyingi ishga o'tib ketish imkonim bor.

Paralellikka nima kiradi? Paralellikka **bir nechta oqimlarga ajratish (multithreading)** va **bir nechta protsessorlarga ajratish(multiprocessing)**larni kiritishimiz mumkin. Lekin yuqorida sanab o'tilgan atamalar ya'ni asinxron, multithreading va multiprocessinglar ishlatilish holatiga qarab biz aytgan bo'lim, ya'ni konkurensiya yoki paralellikka kirmasligi yoki kirishi mumkin. Qisqasi, ishlatilish holati yanada bu narsani aniqlashtiradi.

**Sinxron** kodimiz o'sha biz bilgan har doimgi yozadigan php kodimizdir. PHP kodimizni o'zi emas albatta.

— Bir narsalar deb shuncha boshimni qotirding humpar...

Uzr, berilib ketibman 😊 Lekin yana bir-ikkita mavzu shunday nazariyalarni o'rganamiz. So'kib tashlamangizlar 😊

## Oqim (thread)

Oqimlar haqida tushunchaga ega bo'lsangiz kerak. Agar unday bo'lmasa, hozir ham qisqacha tushuntirib ketaman. Bizlar har bir narsaga yuzaki qarashga o'rganib qolganmiz. O'rganishda ham, o'qishda ham, ishlashda ham... Agar har bir qilayotgan, ko'rayotgan, gapirayotgan gaplarimizga yuzaki qaramaganimizda katta ishlarni bajargan bo'lar edik...

Xo'p, mayli. Har bir narsani o'rganganda, iloji boricha uni mohiyatini tushunishga harakat qilishimiz kerak. Shu jumladan, hozirgi **oqimlar (threads)** ni ham.

**Men hali bular haqida bilmagan narsalarim juda ham ko'p.  
Chuqur mavzular. Lekin o'zim bilganlarimni ularashishga harakat qilaman. Kitobda yozayotgan har bir gaplarimni to'g'rilingini qayta-qayta tekshirib, uni tekshirish uchun vaqt sarflab keyin yozmoqdamon. Ammo inson xatolardan holi emas, shuning uchun kaminaning xatosini ko'rib qolsangiz, uni bizga yetkazishingizdan mamnun bo'lar edik!**

**Oqimlar** — bir nechta bir-biriga o'xshash, bir-birini ma'lumotiga bog'liq bo'lмаган vazifalarni bir vaqtning o'zida birga bajarish uchun juda ajoyib yechimlardan biri. Masalan, siz 10 ta saytdan kerakli ma'lumotlarni yuklab olmoqchisiz. Aytaylik, o'yin sotadigan sayt. U yerda minglab o'yinlar bor, har bir saytni analiz (parse) qilish uchun 1 soatdan vaqt ketyapti. Endi, 10 ta saytni analiz qilsangiz, 10 soat vaqt ketadi. Bu juda katta vaqt. Bu vaqtning 10 barobarga kamaytirishga nima deysiz? Unda, oqimlardan foydalaning. Oqimlardan foydalansak, 10ta saytni ham 10 ta oqimga berib yuboramiz va o'zimiz asosiy oqimda turamiz. Asosiy oqimda turishimizdan maqsad, 10ta oqimdan qaytagan natijani hammasini ko'rsatish. Endi shu 10ta saytimiz 1 soat-u 10 yoki 20 minutlarga, taxminan, cho'zilishi mumkin. Lekin siz 10 soat yutqazmayapsiz. Lekin 10ta ishni natijasi bitta umumiylar narsaga bog'liq bo'ladigan bo'lsa, unda muammo bo'lishi mumkin. Ya'ni, kim o'zar (**race condition**) holatlari paydo bo'lishi mumkin. Bu qanaqa holat? Bu holatda 10 ta oqim ham bitta umumiylar resursga kirishga harakat qiladi.

Lekin resursni natijasi oqimlarning ketma-ketligiga bog'liq bo'lsa, kutilmagan natijaga olib keladi. Oqimlar bilan ishlashda, shu holatlarni inobatga olib qo'yasiz.

## Oqimlarning turlari va ularning ishlashi

Oqimlar turlarga bo'linadi:

1. **Hamkor oqimlar (co-operative threads)**
2. **O'zgaruvchan oqimlar (pre-emptive threads)**

Oqimlar turlarga bo'linishini bilib oldik. Lekin ular qanday ishlaydi? Qaysi biri nima vazifani bajaradi? Hozir shu savollarga javob olamiz.

**O'zgaruvchan oqimlar** — bunday oqimlar operatsion tizim tomonidan boshqarilib, bu oqimlarni to'xtatish yoki rejalashtirish faqat tizim tomonidan amalga oshiriladi. 10 ta oqim ishga tushganda, ularga prioritet belgilanadi. Masalan:

1. 1-oqim
2. 2-oqim
3. ...
4. 10-oqim

Va bu prioritetlar tizimning rejalashtiruvchi dasturi (**scheduler**) tomonidan amalga oshiriladi. Bu rejalashtiruvchi bunday turdag'i oqimlarni har vaqt oralig'ida (**time slice**) prioritetini tegishli tartibda o'zgartirib turadi. Masalan, har **1ms** yoki **10msda**. Qaysi oqimning prioriteti yuqori bo'lsa, o'sha oqim birinchi bajariladi. Shuning uchun, bunday oqimlar o'zgaruvchan oqimlar deyiladi. Ularning tartibi, ustuvorligi o'ziga bog'liq emas va ular istalgan payt tizim tomonidan to'xtatilishi mumkin. Rejalashtiruvchi dastur prioritetlarni bir nechta algoritmlarga qarab belgilaydi. Agar sizga rejalashtirish algoritmlari qiziq bo'lsa, wikipediaga kirib batafsil o'qib ko'rishingiz mumkin ([https://en.wikipedia.org/wiki/Scheduling\\_\(computing\)#SCHEDULER](https://en.wikipedia.org/wiki/Scheduling_(computing)#SCHEDULER)). Chunki men u algoritmlar bo'yicha yaxshi ma'lumotga ega emasman.

Bir oqimdan ikkinchi oqimga o'tishda kontekst har doim almashadi. Bu narsa ingliz tilida **context switch** — **kontekst almashinushi** deb ataladi. Birinchi oqimdan ikkinchi oqimga o'tganda, birinchi oqimning ma'lumotlarini, holatlarini saqlab turishi va qayta unga

murojaat qilinganda, saqlangan ma'lumotlarni qayta tiklashi kerak bo'ladi. Bu esa murakkabroq jarayon bo'lganligi bois, jarayonni sekinlashtirishi mumkin.

Hozir sizlar bilan o'zgaruvchan oqimlar haqida gaplashdik. Yuqorida (10 ta sayt bilan bog'liq namuna) aytilgan oqim, **o'zgaruvchan oqimlar** turiga kiradi.

**Hamkor oqimlar** — bunday oqimlar dasturning o'zi tomonidan istalgan payt to'xtatilishi va ishga tushirilishi mumkin. Bu turdag'i oqimlarga operatsion tizim aralashmaydi. Qaysi dastur to'xtatadi deb o'ylaysiz? Albatta, o'zingiz ishlata yotgan dasturlash tilida qilingan dastur-da!

Bu turdag'i oqimlar bilan siz bir nechta vazifalarni amalga oshirishingiz mumkin. Bu narsa **ko'p vazifalilik (multitasking)** deyiladi. Dehqonchasiga tushuntirsam.

Siz oshxonada ovqat qilyapsiz. **Qozonni tozaladingiz, gazni ustiga qo'ydingiz**, keyin **piyoz, kartoshkani oldingiz, suvda yuvdingiz**. Tortmadan, po'stloq archiydiganni va pichoqni oldingiz. **Piyoz va kartoshkani archib, to'g'radingiz**. Keyin **boshqa ishlaringizni qildingiz**.

Har bir qoraga olgan so'zlarim, alohida vazifalar hisoblanadi. Shu vazifalarni siz muayyan vaqt oralig'ida qilyapsiz. Lekin ularning barchasi har xil vazifalardir. Mana shu narsa **ko'p vazifalilik** deyiladi.

Yana soddaroq, juda soddaroq qilib tushuntirsam, **hamkor oqimni sinxron** kod sifatida tushunishingiz mumkin, **o'zgaruvchan oqimni** esa **asinxron** sifatida.

## **Yashil oqim (green thread)**

Oqimlarni o'rGANIB bo'lGANimizdan keyin biz **yashil oqimlarni** tushunishimiz shart bo'lmay qoladi. Chunki, yashil oqimlar shunchaki hamkor oqimlardir. Nomlari sinonim bo'lgan oqim turlari sirasiga kiradi. Hamkor oqimlar ham yashil oqimlar ham operatsion tizim tomonidan emas balki foydalanuvchi (dasturchi) darajasida boshqariladi.

## **Stack nima?**

**Stack** — bu abstrakt ma'lumot turi bo'lib, **LIFO (last in first out)** tartibida ishlaydi. Ya'ni, oxirgi element kirib, birinchi element chiqib ketadi. Buni xuddi ustma-ust turgan 100lab

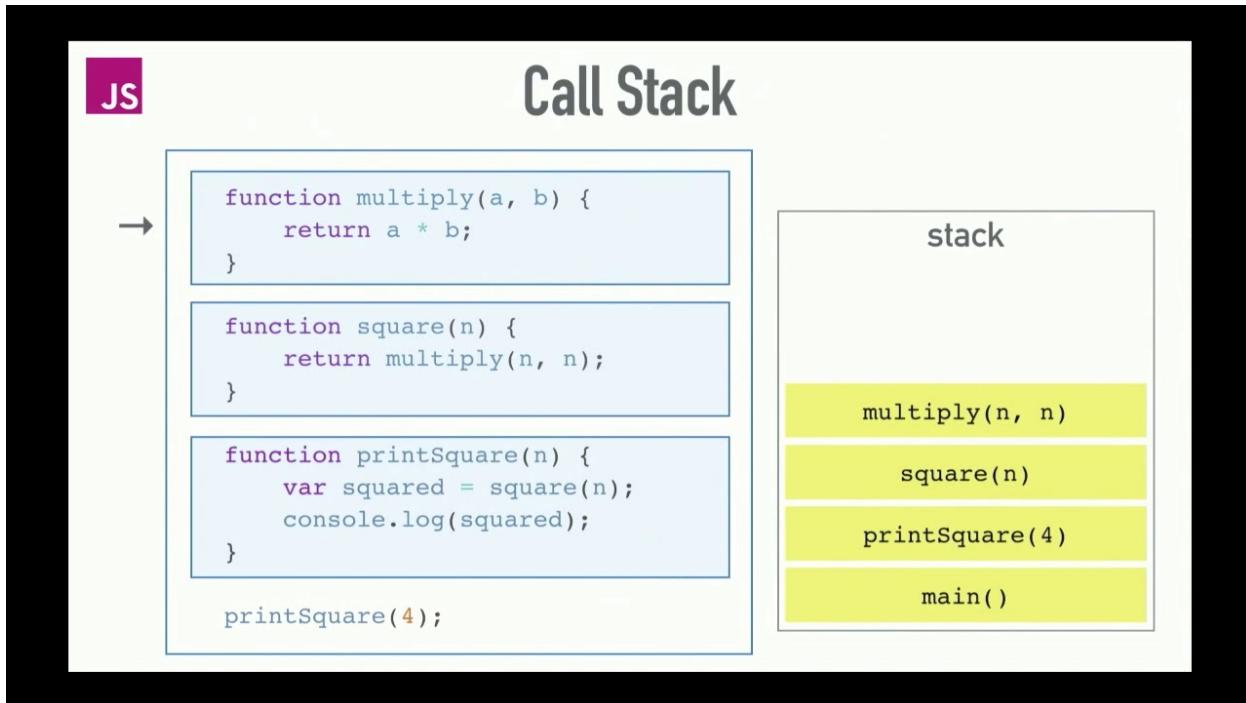
likopchalarga o'xshatish mumkin. Likopchaga qo'shimcha qo'shish uchun o'sha likopchalar ustiga bitta likopcha qo'yib qo'yasiz. Olmoqchi bo'lsangiz, ustidan bittasini olasiz:



Wikipediadan

Hozir oddiy ma'lumot turi haqida gaplashdik. Lekin bu turga o'zimiz istamagan holda har doim duch kelamiz. Chunki, har bir dasturlash tili kompilyatorlari mana shu tur asosida o'ziga kerakli bo'lgan vazifalarni amalga oshiradi. Masalan, ko'pchilikni qulog'iga oz bo'lsa-da chalingan — **call stack**. Bizni dasturimizda nima funksiya ish bajarar ekan, u

**call stack** (keyingi o'rnlarda kolstek) ka yoziladi. Kolstek yuqorida ko'rsatilgan ma'lumot turi kabi ishlaydi:



manba - <https://geekflare.com/javascript-event-loops/> (boshqa yaxshi rasm topolmadim ))

Rasmda ko'rganingizdek, birinchi chaqirilgan funksiya kolstekka birinchi bo'lib yoziladi. O'suvchi tartibda yoziladi va natijalar har doim boshidan pastga qarab ketadi. Xuddi yuqorida **stack** ma'lumot turi ishlash tartibi kabi. Zo'r-a? Tushunarli bo'lgan bo'lsa, unda boshqa mavzuga o'ttik.

## Fiber

**Fiber** (keyingi o'rnlarda **fayber** deyman) — hamkor oqimlar turidan foydalanadi. Bunday turdag'i oqimlar dasturni o'zi tomonidan boshqariladi. Fayberlar dastur tomonidan to'xtatilishi, tugatilishi va davom ettirilishi mumkin.

Fayber o'zini to'xtatganidan so'ng, uni asosiy dastur qayta davom ettirishi kerak aks holda u o'zini davom ettirolmaydi. Fayberlar php tilidagi azaldan mavjud bo'lgan generatorlarga o'xshaydi. Generatorning fayberdan farqi steki va oqimi mavjud emasligida hamda tur qaytarishidadir. Fayberda esa stek va oqim mavjud. Fayberlar turga qaram emas. Fayberlar yashil oqimda ishlaydi. Fayberlarni minglab ichma-ich

funksiyalar ichida ham bermalol to'xtatish mumkin. Lekin generatorlarda bunday imkoniyat yo'q. Chunki generatorlar, qaytarilgan natijaga suyanadi. Ya'ni ular davomli uzatish uslubi bo'yicha ishlaydi. Oddiyroq qilib aytganda, **callback** (kolbek) funksiyalarga o'xshab ketadi. Lekin fayberlar natijaga suyanmaydi:

```
<?php

// ----- GENERATORLAR -----
function mainGenerator(): Generator
{
    for ($i = 0; $i < 10; $i++) {
        yield $i;

        function topLevel (): Generator
        {
            function oneLevel(): Generator
            {
                function twoLevel (): Generator
                {
                    yield 'from hell';
                }

                yield from twoLevel();
            }

            yield from oneLevel();
        }

        yield from topLevel();
    }
}

$gen = mainGenerator();

echo $gen->current();
$gen->next();
echo $gen->current();

// Generator natija: 0from hell

// ----- FAYBERLAR -----
<?php

$fiber = new Fiber(function () {
    for ($i = 0; $i < 10; $i++) {
        Fiber::suspend($i);

        function topLevel (): void
    }
})
```

```

        function oneLevel(): void
    {
        function twoLevel (): void
        {
            Fiber::suspend('from hell');
        }

        twoLevel();
    }

    oneLevel();
}

topLevel();
});

echo $fiber->start();
echo $fiber->resume();

// Fayber natija: 0from hell

```

Ko'rdingizmi? **Fayber** tur qaytarmaydi va natijaga qaram bo'lmaydi. Bu degani **fayber** generatorning yaxshi varianti degani emas. Generatorning vazifasi boshqa, fayber esa boshqa muammo uchun ishlab chiqilgan.

Generatorning vazifasi, asosiy vazifalaridan biri, bunday ichma-ich funksiyalar uchun emas balki funksiya ma'lumotlarini, minglab ma'lumotlarni, xotiraga saqlamasdan turib uni qadamma-qadam generatsiya qilishdan iborat. Bundan maqsad, xotiradan va vaqtdan (sal bo'lsa ham) yutishdir.

A fayberlar esa, hmm.... Hozir o'ylab olay ...

Fayberning asosiy vazifasi diskriminatsiyani yo'q qilish. Diskriminatsiya?! Ha. Aksar joylarda istalgan holat bo'yicha diskriminatsiya bo'lgani kabi, dasturlash sohasida dasturlash tillarida ham bunday illat uchrab turadi 😊 Sinxron va asinxron funksiya.

Sinxron va asinxron funksiya aslida ular bitta funksiyadan iborat lekin sinxron funksiyani ichida asinxron funksiyani ishlatischning iloji yo'q. Ya'ni, asinxron dasturlashni qo'llab-quvvatlaydigan ko'p dasturlash tillarida shunday ajratishlar bor. Shuning uchun, ko'pincha, "*Funksiyang asinxronmi yoki sinxron?*", "*Nega asinxron funksiyani sinxron funksiyaning ichida ishlatyapsan?*", "*Asinxron funksiyang **Promise** qaytarishi kerak-ku!*" kabi dag'dag'a va savollar uchrab turadi...

Va avval aytganimdek, **asinxron** funksiya ham, **generator** ham, hamma hammasi avvalgi qardoshining natijasiga qarab ish tutishadi. Ya'ni, ularning hammasi **davomiy uzatish uslubi** yoki sodda qilib aytganda **callback hell** kabi ishlashadi.

A fayberlarga esa farqi yo'q. Istalgan joyda o'z ishini ortiqcha yuklanishlarsiz bajaradi xolos. Unga na avvalgi funksianing natijasi na keyingi funksianing natijasi kerak.

Bundan xulosa qilishimiz mumkin-ki, fayberlar **multitasking** va **asinxronlik\*** tomon ilk qo'yilgan qadamdir. Ular **asinxron amallar\*** ni bajarish uchun quyi darajadagi dasturiy vosita hisoblanadi.

**Asinxronlik yoki asinxron amallar deganda, biz bilgan (async / await kabi) asinxron funksiyalar nazarda tutilmayapti.**

## JIT (Just In Time) kompilyatsiya

**JIT** ning vazifasi **opkodlarni\*** mashina tiliga o'girishdan iborat. Opkodlarni mashina tiliga o'girish jarayoni kodni ish vaqtida amalga oshirilgani uchun **Just In Time** deb atalgan.

Opkodni quyidagi ko'rinishdagidek tasavvur qilishingiz mumkin:

```
<?php

$opcodes = [
    'ASSIGN' => '=',
    'VALUE' => 5,
    'VARIABLE' => 'd',
    'SEMICOLON' => ';'
];

// Misol uchun bu $d = 5; kodni opkodlari. Bu shunchaki namuna uchun yozilgan.
```

**Virtual mashina (Zend PHP)** manba kodi(**biz yozgan php kod**)ni opkodlarga o'girib chiqib, uni JITga beradi. JIT opkodlarni platformangizni xususiyatiga mos bo'lgan mashina tiliga tarjima qilib beradi. Har safar Zend manba kodni opkodga o'girganda bunday opkod keshda mavjud bo'lsa va **JIT** kompilyatsiya qilgan bo'lsa, uni mashina kodini olib **CPU** (Protssessor)ga to'g'ridan to'g'ri yuboradi. Bu esa tezlikni sezilarli ravishda ortishiga sabab bo'ladi.

Zendda **Opcache** vositasi mavjud bo'lib, JIT busiz ishlay olmaydi. Opkeshning vazifasi keshlashdan iborat. Opkesh opkodlarni keshlash uchun ishlatiladi. my.php nomli faylimizning opkodlari bir marta analiz qilib, generatsiya qilinganidan so'ng, agar uni ichida hech qanday narsa o'zgarmagan bo'lsa va o'sha fayl qayta ishlatilish paytida bo'lsa, Zend bu fayl opkeshda keshlanganligini tekshiradi, agar bu fayl keshda topilsa, uning opkodlarini interpretatsiya qiladi.

A JIT esa, shu opkodlarni mashina kodini generatsiya qilib, interpretatsiyaga hojat qoldirmaydi.

JIT funksionali **PHP 8.0** versiyadan boshlab mavjud.

start > rfc > jit

## PHP RFC: JIT

https://wiki.php.net/rfc/jit>. Below the metadata is a section titled 'Introduction' with a paragraph about the history of JIT implementation in PHP. There are also sections for 'The Case for JIT Today' and 'Proposal' with their respective content."/>

- Version: 1.0
- Date: 2019-01-28
- Author: Dmitry Stogov <[dmitry@php.net](mailto:dmitry@php.net)>, Zeev Suraski <[zeev@php.net](mailto:zeev@php.net)>
- Status: Implemented (PHP 8.0)
- First Published at: <<https://wiki.php.net/rfc/jit>>

### Introduction

It's no secret that the performance jump of PHP 7 was originally initiated by attempts to implement JIT for PHP. We started these efforts at Zend (mostly by Dmitry) back in 2011 and since that time tried 3 different implementations. We never moved forward to propose to release any of them, for three main reasons: They resulted in no substantial performance gains for typical Web apps; They were complex to develop and maintain; We still had additional directions we could explore to improve performance without having to use JIT.

### The Case for JIT Today

Even though most of the fundamentals for JIT-enabling PHP haven't changed - we believe there is a good case today for JIT-enabling PHP.

First, we believe we've reached the extent of our ability to improve PHP's performance using other optimization strategies. In other words - we can't further improve the performance of PHP unless we use JIT.

Secondly - using JIT may open the door for PHP being more frequently used in other, non-Web, CPU-intensive scenarios - where the performance benefits will actually be very substantial - and for which PHP is probably not even being considered today.

Lastly - making JIT available can provide us (with additional efforts) with the ability to develop built-in functions in PHP, instead of (or in addition to) C - without suffering the huge performance penalty that would be associated with such a strategy in today's, non-JITted engine. This, in turn, can open the door to faster innovation - and also more secure implementations, that would be less susceptible to memory management, overflows and similar issues associated with C-based development.

### Proposal

<https://wiki.php.net/rfc/jit>

JIT ilk bor **2016-yil** taklif etilgan. Qarang, oradan 7 yil vaqt o'tganidan so'ngina bu narsa qo'shildi. Tasavvur qilyapsizmi?

**JIT for PHP project**

**From:** [Dmitry Stogov](#)    **Date:** Thu, 01 Sep 2016 11:57:24 +0000  
**Subject:** JIT for PHP project  
**Groups:** [php.internals](#)

Hi @internals,

I'm glad to say that we have started a new JIT for PHP project and hope to deliver some useful results for the next PHP version (probably 8.0). We are very early in the process and for now there isn't any real performance improvement yet. So far we spent just 2 weeks mainly working on JIT infrastructure for x86/x86\_64 Linux (machine code generation, disassembling, debugging, profiling, etc), and we especially made the JIT code-generator as minimal and simple as possible. The current state, is going to be used as a starting point for research of different JIT approaches and their usability for PHP.

The code is available at: <https://github.com/zendtech/php-src/tree/jit-dynasm/ext/opcache/jit>

The sources may be built and tested as regular PHP (no any special external dependencies required). JIT itself is implemented as a part of Opcache. You may try it in action:

```
sapi/cli/php -d opcache.jit_buffer_size=32M Zend/bench.php
sapi/cli/php -d opcache.jit_buffer_size=32M -d opcache.jit_debug=1 Zend/bench.php 2>&1 | less
```

As I mentioned we didn't try to achieve any real performance improvement yet, although we do currently see 20% speedup on bench.php, but a bit of a slowdown on real-life apps.

Wish us luck :)

Thanks. Dmitry.

[« previous](#)    php.internals (#95531)    [next »](#)

<https://news-web.php.net/php.internals/95531>

PHP tilidagi JIT, **LuaJit** tomonidan ishlab chiqilgan **Dynasm** dvijogidan foydalanadi. Bu loyiha haqida bat afsil ma'lumot uchun — <https://luajit.org/dynasm.html>

## Afzalliklari

- Kodning ishlashi sezilarli ravishda tezlashadi
- JITsiz protsessorga katta yuklanish beradigan amallarni JIT bilan osonroq bajariladi

## Kamchiliklari

- JIT Virtual mashinani chetlab o'tganligi bois, xatoliklarni topadigan vositalar (**xdebug** masalan) bilan ishlashda muammolar kelib chiqishi
- Birinchi marta ishlaganda nisbatan ko'proq vaqt olishi
- Qat'iy turlanmagan ma'lumotlar (funksiya, o'zgaruvchi) tufayli sekinlashishi

**PHP** da kod tezroq ishlashi uchun **strict\_types** rejimini yoqib qo'yishni tavsiya qilaman. Bu ham **JIT**ga ham **PHP** kodni o'ziga ijobiy ta'sir qiladi. Ya'ni, siz bu bilan PHP kodni

analiz qilayotgan paytda uni turni o'zi aniqlashga ketkazadigan vaqtni foydali narsaga sarflashiga sababchi bo'lasiz.

#### opcache.jit stringlint

For typical usage, this option accepts one of four string values:

- disable: Completely disabled, cannot be enabled at runtime.
- off: Disabled, but can be enabled at runtime.
- tracing/on: Use tracing JIT. Enabled by default and recommended for most users.
- function: Use function JIT.

For advanced usage, this option accepts a 4-digit integer CRT0, where the digits mean:

##### c (CPU-specific optimization flags)

- 0: Disable CPU-specific optimization.
- 1: Enable use of AVX, if the CPU supports it.

##### R (register allocation)

- 0: Don't perform register allocation.
- 1: Perform block-local register allocation.
- 2: Perform global register allocation.

##### T (trigger)

- 0: Compile all functions on script load.
- 1: Compile functions on first execution.
- 2: Profile functions on first request and compile the hottest functions afterwards.
- 3: Profile on the fly and compile hot functions.
- 4: Currently unused.
- 5: Use tracing JIT. Profile on the fly and compile traces for hot code segments.

##### o (optimization level)

- 0: No JIT.
- 1: Minimal JIT (call standard VM handlers).
- 2: Inline VM handlers.
- 3: Use type inference.
- 4: Use call graph.
- 5: Optimize whole script.

The "tracing" mode corresponds to CRT0 = 1254, the "function" mode corresponds to CRT0 = 1205.

<https://www.php.net/manual/en/opcache.configuration.php#ini.opcache.jit>

Bu parametrlarini o'zim hali aniq bilmayman. Chunki juda chuqur mavzular bular.

Shuning uchun, eng optimal varianti — 1255 yoki "tracing" variantini ishlattim.

opcache.jit o'zgaruvchiga yoki rasmdagi 4ta satrli parametrlardan birini yoki moslashuvchan sonli parametrlarni berishingiz mumkin. Agar raqamdan foydalansangiz,

4ta xonali son belgilashingiz kerak. Bu yoki [1255](#) yoki ["tracing"](#). Aynan shu qiymat, JITning bor imkoniyatidan foydalanishga yordam beradi.

```
# JIT konfiguratsiyalari

opcache.enable=1
opcache.enable_cli=1
opcache.jit_buffer_size=256M
opcache.jit=1255
```

`opcache.jit_buffer_size` o'zgaruvchisi keshda qancha hajmdagi **JIT** kompilyatsiya qilgan mashina kodlari saqlanishi mumkinligini ifodalaydi.

## Tezlik raqamlarda

**JIT** va **Opcache** o'chiq holatda:

```
empty_loop 0.045 func() 0.089 0.044 undef_fun
0.036 Foo::f() 0.080 0.034 $x = $this->x 0.053 (
Foo() 0.132 0.086 $x = TEST 0.052 0.007 $x =
```

**Total 2.351**

**JIT** o'chiq, **Opcache** yoniq holatda:

```
empty_loop 0.049 func() 0.033 -0.016 undef_func() 0  
-0.019 Foo::f() 0.030 -0.019 $x = $this->x 0.053 0.004  
new Foo() 0.131 0.083 $x = TEST 0.053 0.005 $x = $
```

**Total 1.744**

JIT va Opcache yoniq holatda:

```
empty_loop 0.008 func() 0.008 0.000 undef_func() 0  
-0.007 Foo::f() 0.002 -0.007 $x = $this->x 0.006 -0.001  
Foo() 0.082 0.073 $x = TEST 0.005 -0.003 $x = $_G
```

**Total 0.598**

Ko'rib turganingizdek, tezlik sezilarli ravishda oshgan garchi bu kichik sonlar bo'lsa ham. Bu benchmark ushbu kod repozitorisi orqali tekshirildi — [https://github.com/php/php-src/blob/php-8.0.0/Zend/micro\\_bench.php](https://github.com/php/php-src/blob/php-8.0.0/Zend/micro_bench.php)

## FFI (Foreign Function Interface)

**PHP** dasturlash tilida **C** funksiyalaridan foydalanish mumkinligini bilarmidингиз? Agar sizda tayyor **C** tilida yozilgan dastur bo'lsa lekin uni **PHP** da ham ishlatmoqchi bo'lsangiz, buni amallasangiz bo'ladi. Buning uchun **FFI** funksionalidan foydalanishingiz kerak.

Bu funksiyadan foydalanish uchun manba:

- <https://www.php.net/manual/en/ffi.examples-complete.php>
- <https://www.php.net/manual/en/ffi.examples-basic.php>

O'zingiz ko'rib chiqsangiz bo'ladi. Bu shunchaki ma'lumot sifatida keltirildi. **C** tilini bilmaganim va ishim tushmagani uchun hozircha bu narsa kerak bo'lgani yo'q.

## Yakun

Men siz bilan bo'lishmoqchi bo'lgan narsalarim shulardan iborat edi. Agar kitobda kamchilik topgan bo'lsangiz, iltimos menga xabar qiling. Sizning murojaatingiz men uchun qadrli!

**Xurmat bilan,**

Sanjarbek Sobirjonov.

2023-yil, 4-mart.

Toshkent sh.