

# Wstęp do programowania

v.1.2

# Plan

- [Trochę teorii](#)
- [Czas na praktykę](#)
- [Podstawowa składnia PHP](#)
- [Dane w programie](#)
- [Kontrola przepływu programu](#)
- [Funkcje](#)

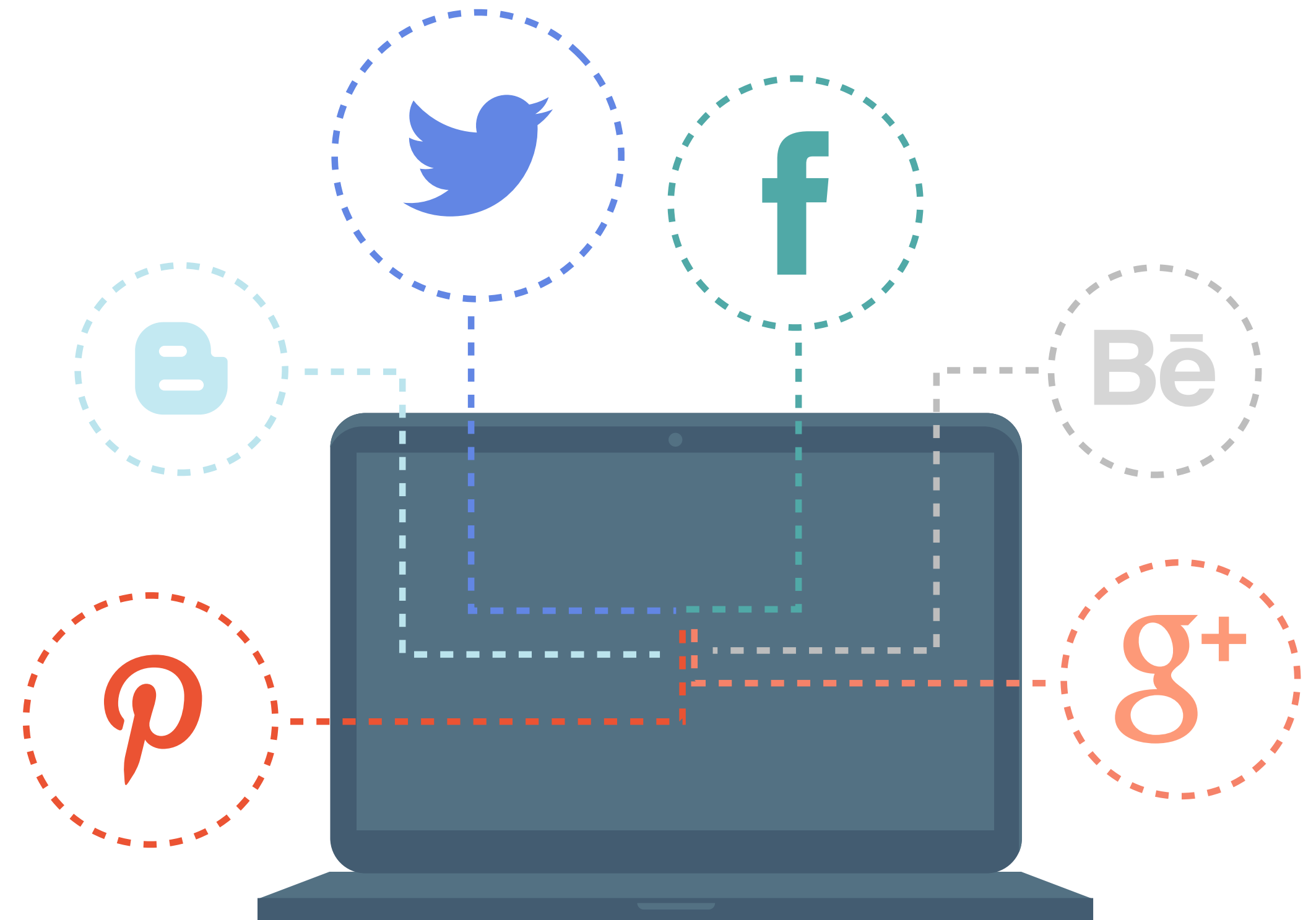
- [Dobre praktyki](#)
- [Najczęstsze błędy początkujących](#)
- [Linki przydatne każdemu programiście](#)



**Trochę teorii**

# Co nazywamy komputerem?

Współczesne komputery od wszystkich innych maszyn odróżnia **możliwość ich programowania**, czyli wprowadzenia do pamięci komputera listy instrukcji, **które mogą być wykonane w innym czasie**.



# Język programowania i kod źródłowy

## Język programowania

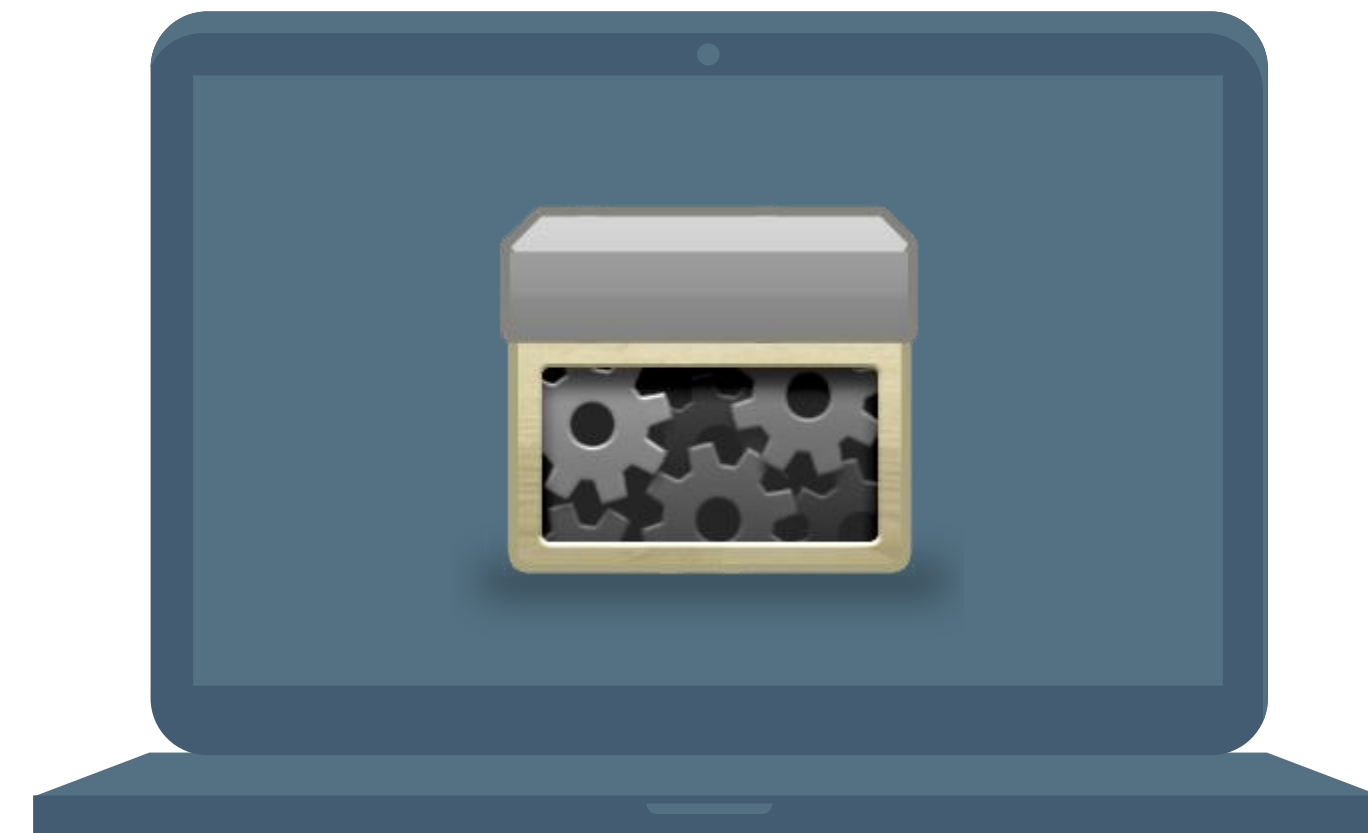
- Jest to zbiór zasad semantycznych, wskazujących, jak pisać tekst zrozumiały zarówno przez człowieka, jak i przez komputer.
- Język programowania często zawiera bibliotekę najczęściej używanych funkcji. Dzięki temu wykonanie wielu zadań jest łatwiejsze.

## Kod źródłowy

- Jest to algorytm (lub wiele współpracujących algorytmów) rozwiązujący dany problem.
- Kod jest zapisany w konkretnym języku programowania.
- Zapis ten jest czytelny dla człowieka.

# Program

Program to wynik kodu źródłowego, który jest tłumaczony (przez inny program np. kompilator lub interpreter) na kod maszynowy zrozumiały przez komputer i gotowy do uruchomienia.



# Język programowania a język znaczników

Nie każdy język zrozumiały dla komputera jest językiem programowania.

Wyróżniamy jeszcze **języki znaczników**.

Są to języki:

- opisujące wygląd przechowywanej treści,
- nie da się w nich realizować obliczeń i kontrolować przepływu treści,
- najpopularniejszym językiem znaczników stosowanym obecnie jest **HTML**.

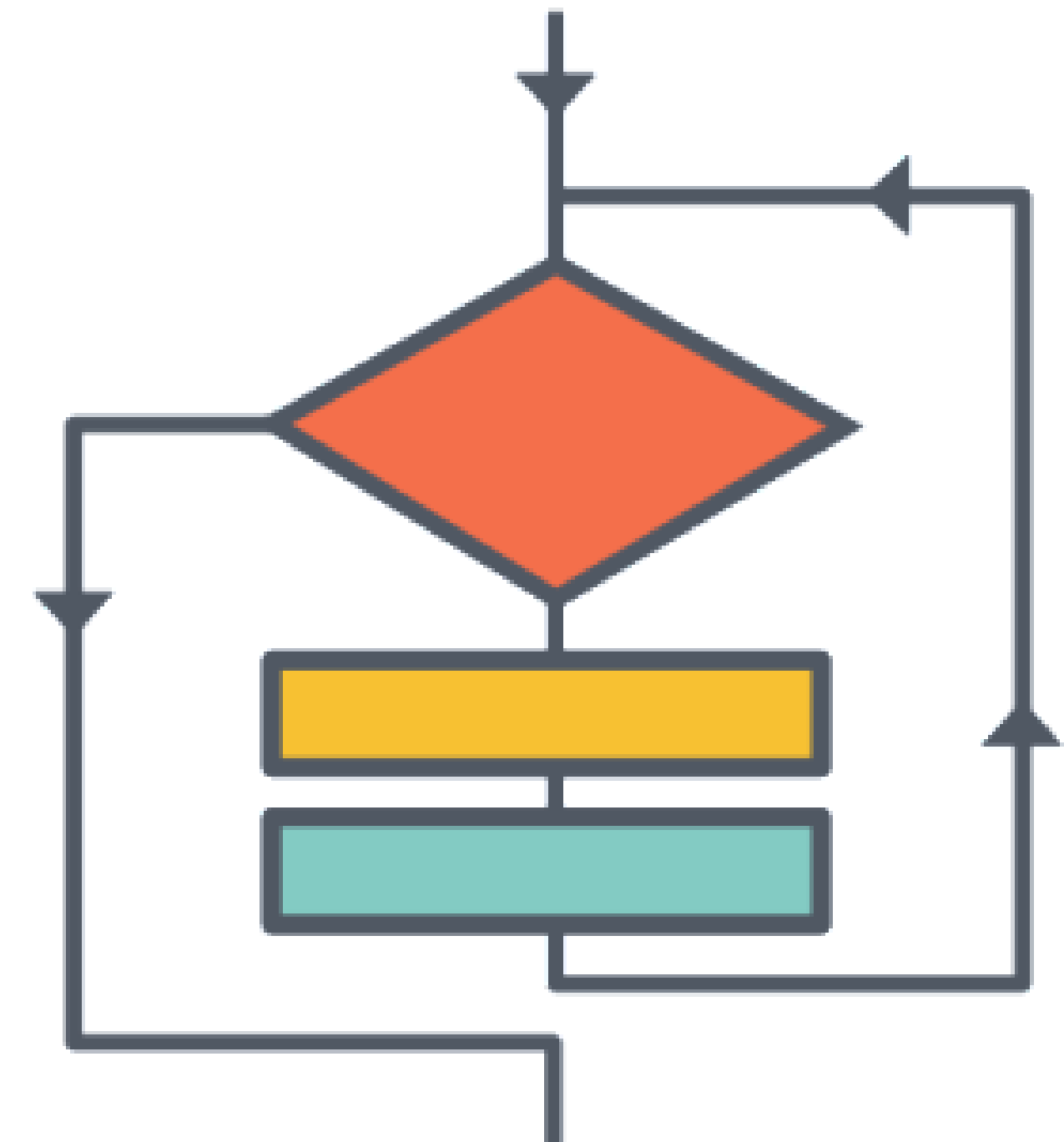


# Co to jest algorytm?

Algorytm – skończony ciąg jasno zdefiniowanych czynności koniecznych do wykonania pewnego rodzaju zadań.

Gdzie na co dzień spotykamy się z algorytmami?

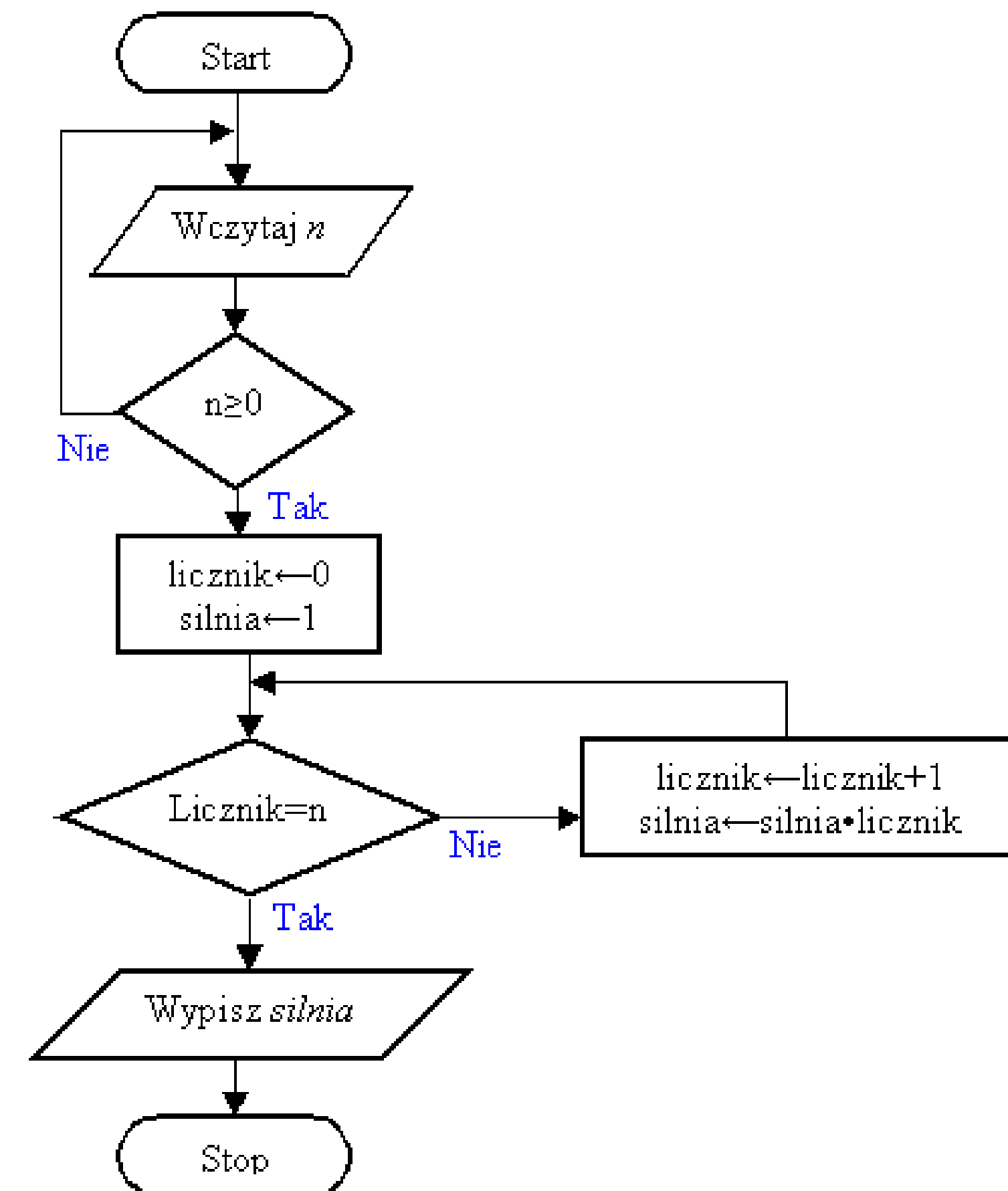
- W kuchni – wszystkie przepisy.
- Na drogach – zasady ruchu drogowego.
- Praktycznie w każdej dziedzinie naszego życia.





# Schemat blokowy (flowchart)

- **Strzałka** – wskazuje jednoznacznie powiązania i ich kierunek.
- **Prostokąt** – zawiera wszystkie operacje z wyjątkiem instrukcji wyboru.
- **Równoległobok** – wejście/wyjście danych.
- **Romb** – wpisujemy wyłącznie instrukcje wyboru,
- **Owal** – oznacza początek bądź koniec schematu.



# Czas na zadania

- Na swoim koncie GitHub powinniście mieć udostępnione repozytorium o nazwie Podstawy\_PHP.
- Przeróbcie grupowo ćwiczenia z części A.  
Ćwiczenie numer A1 zostanie rozwiązane przez wykładowcę.

# Pseudokod

## Definicja

- Pseudokod to popularna notacja, w której zapisywane są algorytmy.
- Składa się z instrukcji sterujących, zmiennych przypisanych. Często też z opisu tekstowego w języku naturalnym.
- Nie ma bardzo sztywnych reguł.

## Przykład

```
funkcja foo(a, b)
    dopóki a ≠ b
        jeżeli a > b
            a = a - b
        inaczej
            b = b - a
    zwróć a
```

# Algorytmy są niezależne od języka

- Algorytm to idea działania programu.
- Algorytm idealny powinien być zapisany w pseudokodzie.
- Pseudokod algorytmu można przełożyć na praktycznie każdy język programowania.



# Porównanie języków

PHP	C++	RUBY	Co robi?
<code>print("cześć");</code>	<code>printf("cześć");</code>	<code>print "cześć"</code>	Wpisuje na ekranie słowo <b>cześć</b>
<code>\$arr = array(3,2,1);</code>	<code>int arr[ ] = {3,2,1};</code>	<code>arr = [3,2,1]</code>	Tworzy tablicę z danymi: 3,2,1
<code>sort(\$arr);</code>	<code>std::sort (arr, arr + 3);</code>	<code>arr.sort</code>	Sortuje podaną tablicę z danymi
<code>if (warunek) {     operacja; }</code>	<code>if (warunek) {     operacja; }</code>	<code>if warunek     operacja end</code>	Wykona daną operację, o ile warunek zostanie spełniony



**Czas  
na  
praktykę!**

# Pierwszy program w PHP

```
<?php
echo('Hello World');
?>
```

Linia zaczynająca każdy skrypt PHP.

Komenda **echo** powoduje wypisanie podanego tekstu na ekranie.

Linia kończąca każdy skrypt PHP.

# Co w razie błędu?

- Jeśli w naszym skrypcie jest błąd składniowy, **skrypt będzie wykonywany**, aż natrafi na ten błąd!
- W przypadku błędu podane zostaną następujące informacje:
  - typ błędu,
  - plik, w którym ten błąd wystąpił,
  - linia zawierająca błąd.

## Kod PHP

```
<?php
```

```
echo('Hello World');
```

```
echo_error('Hello World');
```

```
?>
```

Ta linia kodu została wykonana

W tej linii jest błąd

## Wynik na stronie

(!) Fatal error: Call to undefined function echo_error() in /var/www/html/error_test/index.php on line 14				
Call Stack				
#	Time	Memory	Function	Location
1	0.0012	126632	{main}()	../index.php:0

Opis błędu

Call stack, czyli dokładny opis miejsca w którym był nasz program w chwili błędu

Dokładny numer linii z błędem




# Błąd a ostrzeżenie

W przypadku błędu interpreter PHP nie jest w stanie dalej działać. Nasz program przestaje się wykonywać

## Kod PHP

```
<?php
    echo('Hello World');
    echo_error('Hello World');
?>
```

## Wynik na stronie


 Fatal error: Call to undefined function echo_error() in /var/www/html/error_test/index.php on line 14				
Call Stack				
#	Time	Memory	Function	Location
1	0.0012	126632	{main}()	../index.php:0

W przypadku ostrzeżenia (częściej nazywanego **notice** albo **warning**) interpreter PHP dostrzega potencjalnie błędną sytuację, w której dalsze działanie jest możliwe.

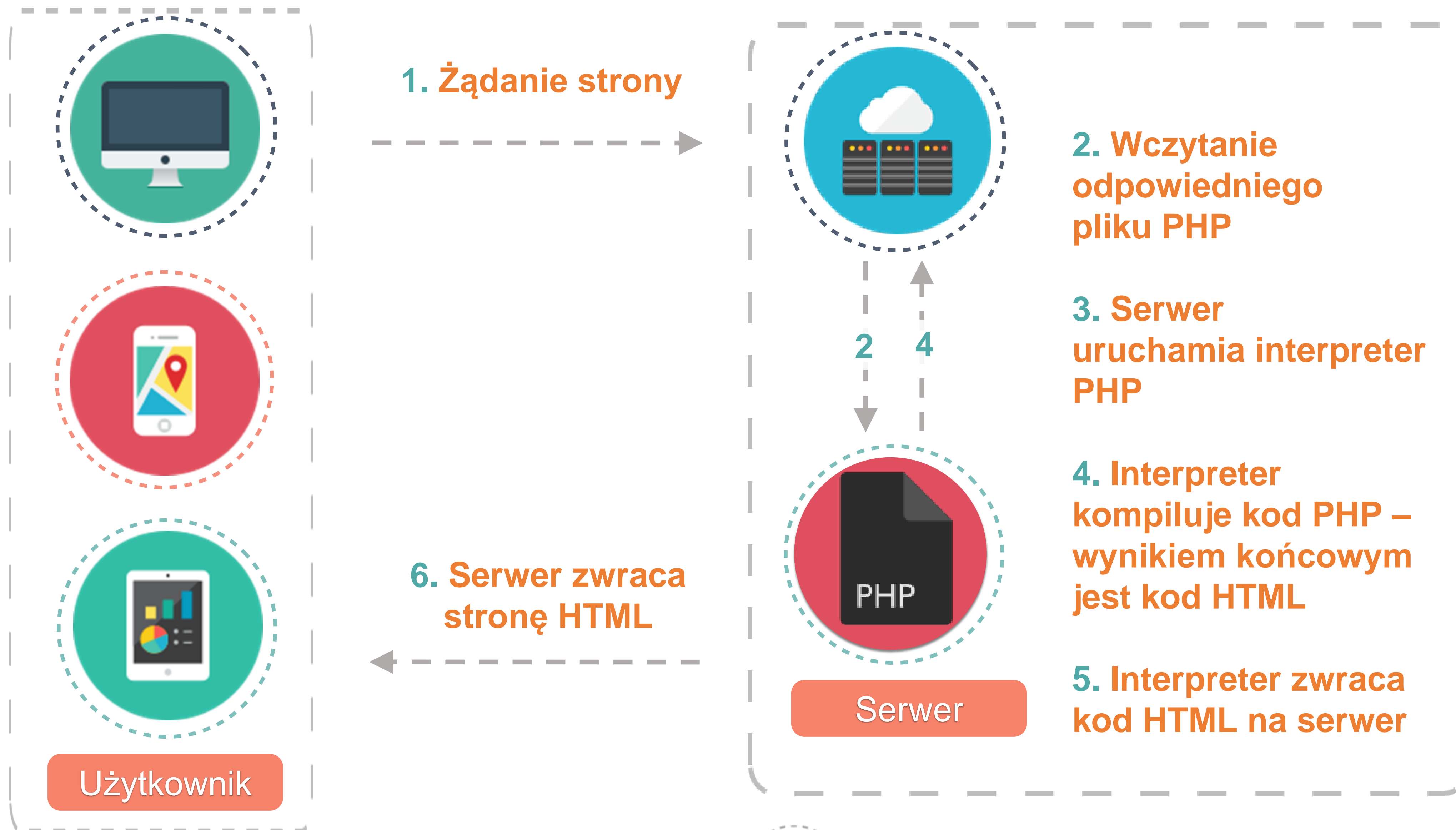
## Kod PHP

```
<?php
    $foo;
    $bar = 4 + $foo;
    echo( $bar );
?>
```

## Wynik na stronie

 Notice: Undefined variable: foo in /var/www/html/test_2/index.php on line 15				
Call Stack				
#	Time	Memory	Function	Location
1	0.0024	125388	{main}()	../index.php:0

# Jak działa serwer PHP?



# Jak działa interpreter PHP?

- Interpreter PHP może działać w dwóch trybach:
  - HTML,
  - PHP.

Dlatego w plikach PHP możemy pisać za równo kod PHP jak i HTML.

- Interpreter zawsze zaczyna w trybie HTML i przełącza się odpowiednio przy znacznikach.
- W trybie HTML interpreter kopiuje tekst do pliku wynikowego.
- W trybie PHP interpreter uruchamia kod PHP. W tym trybie możemy pisać do kodu HTML, używając odpowiednich funkcji.

## Kod

```
<body>  
<h1>Nagłówek</h1>
```

Dokument HTML

Kod pisany w PHP

```
<?php  
    echo('Część dokumentu HTML, którą wygeneruje PHP.');
```

```
    echo('<br/><b>Możemy użyć znaczników HTML.</b><br/>');
```

```
?>
```

Ciąg dalszy dokumentu HTML

```
<p>Paragraf</p>  
</body>
```

## Plik wynikowy

# Nagłówek

Część dokumentu HTML, którą wygeneruje PHP.  
**Możemy użyć znaczników HTML.**

Paragraf

# Podstawowa składnia PHP

# Oddzielanie instrukcji

- Każda instrukcja PHP musi kończyć się znakiem średnika.
- Instrukcja może mieć więcej niż jedną linię.



`echo('Hello')`

Błąd (brak średnika na końcu wyrażenia).

`echo('Hello');`

Poprawnie.

`echo('`

`Hello');`

Też poprawnie.

# Komentarze

- W kodzie PHP można dodać teksty, które będą ignorowane przez interpreter.
- Są to tak zwane **komentarze**.
- Komentarzy używamy zarówno do opisywania, co robi kod (dla innych programistów), jak i chwilowego wyłączania niepotrzebnych części skryptu.

# Znak hash "#" komentuje wszystkie znaki do końca linii.

// Tak samo działa podwójny ukośnik.

/\*

Między tymi znacznikami wszystko jest zakomentowane.

\*/



**Dane w programie**



W JAKI SPOSÓB  
KOMPUTER  
PRZECHOWUJE DANE?



# W jaki sposób komputer przechowuje dane?

01000001 01000010 01000011



ABC

Wszystko w komputerze  
jest liczbą zapisaną binarnie,  
ale nie wszystkie  
takie same liczby znaczą to  
samo.



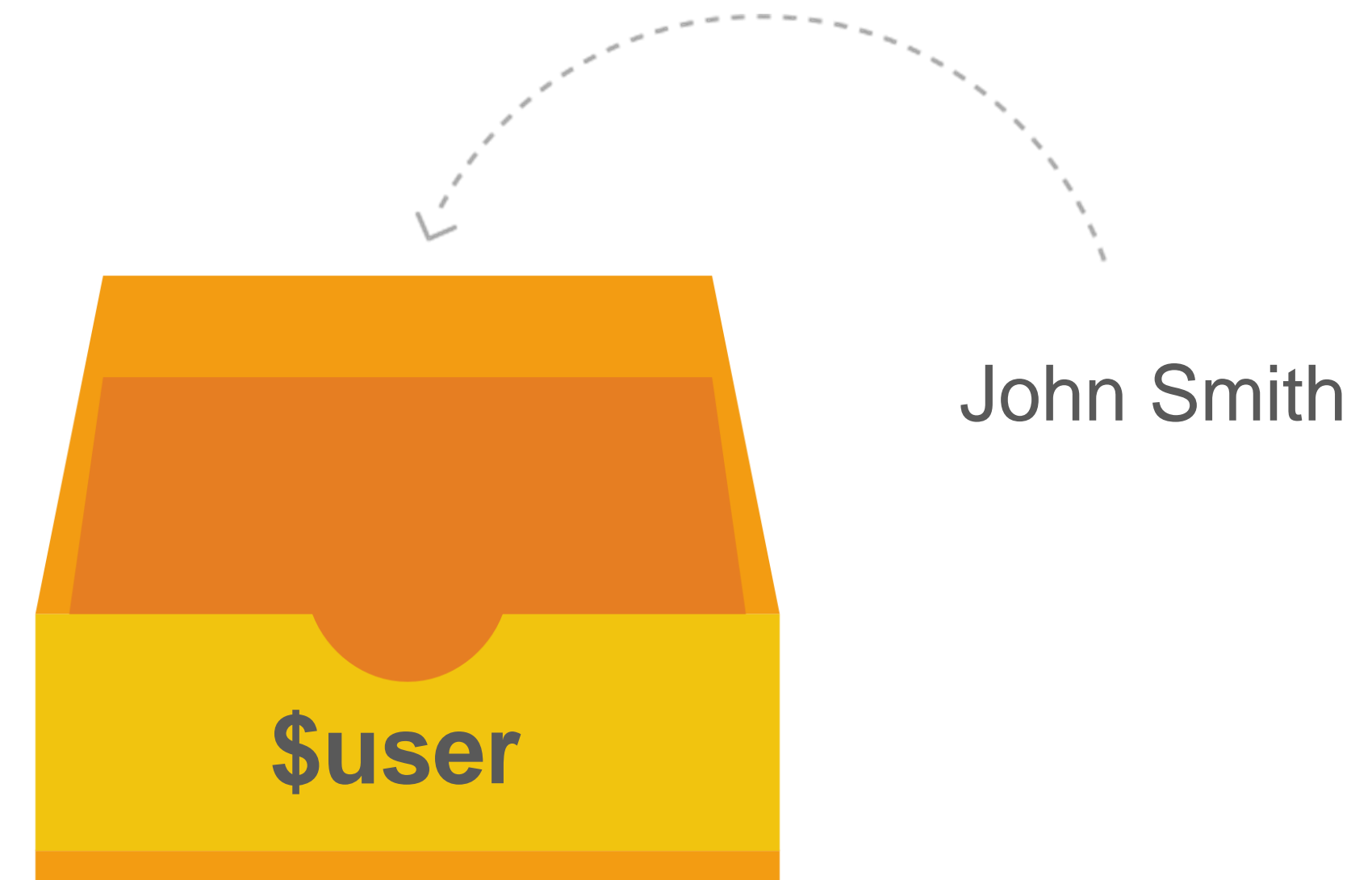
rgb(65,66,67)

# Dane w programie – zmienne

- Zmienne to podstawowy sposób trzymania danych w programie.
  - Zmienna w pewnym uproszczeniu to etykieta odnosząca się do jakiegoś obszaru pamięci.
- Zmienne mają swój typ (może on być na stałe przypisany do etykiety lub nie – mówimy odpowiednio o **statycznym** bądź **dynamicznym** typowaniu).

# Dane w programie – zmienne

- Zmienna to tak naprawdę pudełko, które coś trzyma.
- Takie pudełko ma na sobie nalepkę ze swoją nazwą.



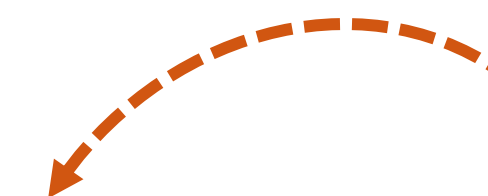
# Zmienne – zasady

## Nazwa zmiennej

- Każda nazwa zmiennej musi być poprzedzona znakiem \$.
- Składa się z dowolnej liczby liter, cyfr i niektórych znaków specjalnych.
- Nazwa zmiennej musi zaczynać się od litery.

## Dwie różne zmienne

\$test = 1;  
\$TEST = 12;

 To są dwie różne zmienne.

**Uwaga! Zmienne są czułe na wielkość znaków.**

# Kontrola przepływu programu

# if oraz if razem z else

Tę instrukcję warunkową można porównać do obecnego w języku naturalnym stwierdzenia:

jeśli (**if**)..., to zrób....,

jeśli nie poprzednie ale (**else if**) ..., to zrób ...

jeśli nie wszystkie poprzednie (**else**), to zrób....

Część **else if** i **else** jest opcjonalna.

```
if(wyrażenie_warunkowe) {  
    //instrukcja wykonywana,  
    //jeśli spełniony zostanie warunek  
}  
else if(inne_wyrażenie_warunkowe) {  
    //instrukcje wykonywana, jeśli spełniony  
    //zostanie drugi warunek, a pierwszy nie  
}  
else {  
    //instrukcja wykonywana, jeśli nie zostanie  
    //spełniony żaden z poprzednich warunków  
}
```

# Czas na zadania

- Przeróbcie ćwiczenia z części B.  
Ćwiczenie numer B1 zostanie rozwiązane przez wykładowcę.

# Pętla while

Najprostsza pętla.

Można ją porównać do stwierdzenia:

dopóki (**while**) \_\_\_\_\_,  
powtarzaj \_\_\_\_\_.

```
while(warunek) {  
    //instrukcje do wykonania w pętli  
}
```

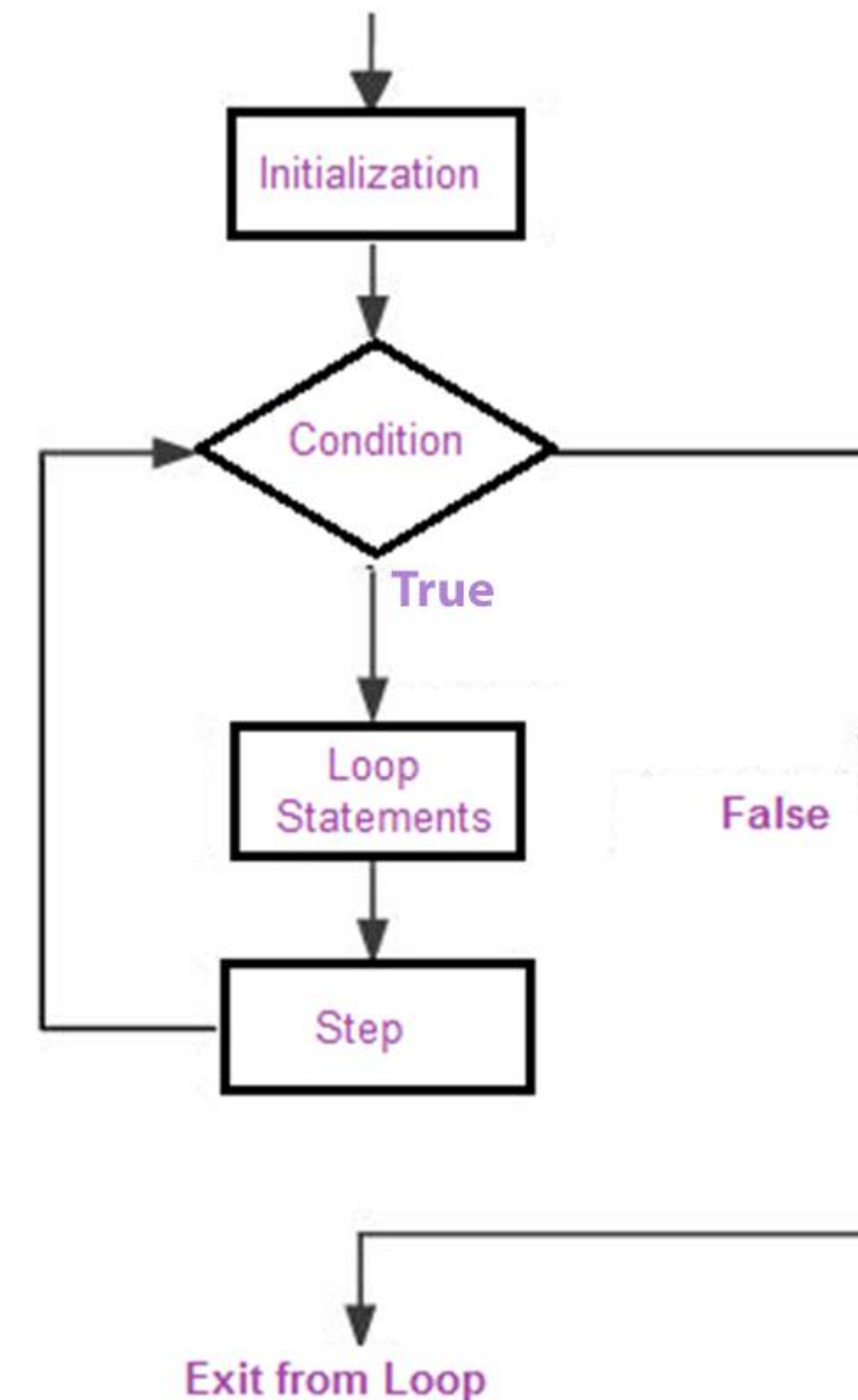


# Pętla for

Bardziej skomplikowana pętla  
(ale najpopularniejsza w programowaniu).

-----

```
for(inicjalizacja zmiennych;  
    sprawdzenie warunku;  
    modyfikacja zmiennych) {  
    //instrukcje do wykonania w pętli  
}
```



# Pętla for

## Kod PHP

```
for($i=0; $i<=10; $i=$i+1) {  
    echo($i);  
    echo("<br>");  
}
```

## Wynik

0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10

Zwyczajowo zmienne  
które sterują pętlą  
nazywamy \$i, \$j albo  
\$k

## Kod PHP

```
for($i=10; $i>=0; $i=$i-1) {  
    echo($i);  
    echo("<br>");  
}
```

## Wynik

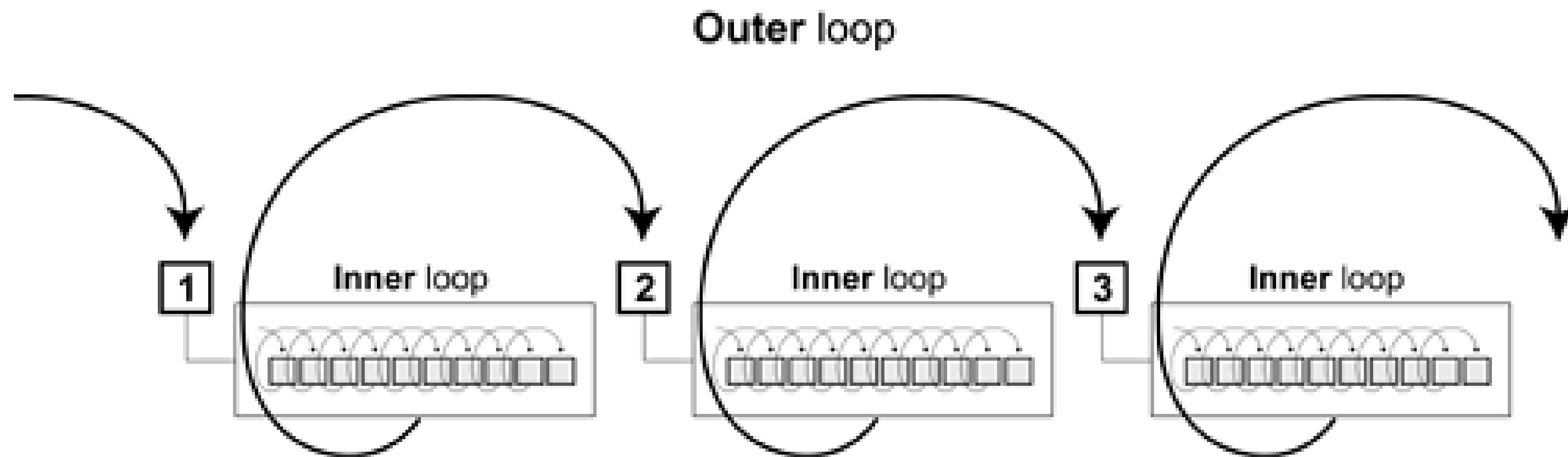
10  
9  
8  
7  
6  
5  
4  
3  
2  
1  
0

# Czas na zadania

- Przeróbcie ćwiczenia z części C.  
Ćwiczenie numer C1 zostanie rozwiązane przez wykładowcę.

# Pętla for (podwójna)

- Pętle możemy w sobie zagnieżdżać.
- Dzięki temu w każdej iteracji pętli zewnętrznej będzie wykonywane wiele iteracji pętli wewnętrznej.



# Pętla for (podwójna – niezależna)

Jeżeli zmienne które sterują pętlami nie wpływają na siebie mówimy o pętlach niezależnych.

```
for($i=0; $i<10; $i=$i+1) {  
    for($j=0; $j<10; $j=$j+1) {  
        echo("i=" . $i . ", j=" . $j . "<br>");  
    }  
}
```

# Pętla for (podwójna – zależna)

Możemy też uzależnić te zmienne od siebie (np. przypisując numer iteracji pętli zewnętrznej jako start pętli wewnętrznej).

```
for($i=0; $i<10; $i=$i+1) {  
    for($j=$i; $j<10; $j=$j+1) {  
        echo("i=" . $i . ", j=" . $j . "<br>");  
    }  
}
```

# Czas na zadania

- Przeróbcie ćwiczenia z części D.  
Ćwiczenie numer D1 i D2 zostanie rozwiązane przez wykładowcę.

# Funkcje



# Funkcje

- Funkcje to odseparowany kawałek kodu wykonujący jakieś działanie i zwracający jakąś wartość.
- Funkcji używamy w celu przejrzystości i możliwości ponownego używania kodu.

## Przykładowa funkcja

```
function sum($x, $y) {  
    $z = $x + $y;  
    return $z;  
}
```

```
$num1 = 20;  
$num2 = 15;
```

```
$numSum = sum($num1, $num2);  
echo("Suma liczb" . $num1 . " i " . $num2 . " = " .  
    $numSum );
```

# Funkcje

```
function sum($x, $y) {  
    $mySum = $x + $y;  
    return $mySum;  
}
```

Deklaracja funkcji zaczyna się od słowa kluczowego **function**, potem wpisujemy nazwę naszej funkcji (nazwy zwyczajowo są po angielsku).

Pomiędzy nawiasami klamrowymi znajduje się **ciało funkcji**, czyli kod który zostanie uruchomiony kiedy użyjemy tej funkcji.

Słowo kluczowe **return** oznacza miejsce w którym nasza funkcja się kończy. Powoduje też zwrócenie podanej wartości do miejsca gdzie wywołaliśmy funkcję.

```
$num1 = 20;  
$num2 = 15;
```

**Wywołanie funkcji** to miejsce gdzie używamy wcześniej napisanej funkcji. W tym miejscu pojawi się jej wynik który zwracamy za pomocą **return**.

```
$numSum = sum($num1, $num2);  
echo("Suma liczb" . $num1 . " i " . $num2 . " = " . $numSum );
```

# Funkcje - argumenty

```
function sum($x, $y) {  
    $mySum = $x + $y;  
    return $mySum;  
}
```

Podczas deklaracji funkcji możemy wskazać też jej **argumenty**. Są to zmienne które trzeba podać podczas jej użycia.

**Argumentów** używamy jak każdej innej zmiennej. Po prostu do chwili użycia nie będziemy znali ich dokładnych wartości.

```
$num1 = 20;  
$num2 = 15;
```

Podczas wywołania funkcji musimy podać **wszystkie argumenty** których funkcja się spodziewa. Mogą to być zarówno nasze zmienne, jak i bezpośrednio wpisane wartości.

```
$numSum = sum($num1, $num2);  
echo("Suma liczb" . $num1 . " i " . $num2 . " = " . $numSum );
```

# Dwa typy funkcji

Funkcje w programowaniu możemy podzielić na dwa typy:

## **Funkcje które nie zwracają wartości:**

Są to funkcje gdzie interesuje nas efekt uboczny wywołania takiej funkcji.

Może być to np.:

- Wyświetlenie wiadomości na ekranie,
- Wysłanie maila przez serwer,
- Zapisanie informacji do bazy danych.

## **Funkcje które zwracają wartość:**

Są to funkcje gdzie interesuje nas zwracana wartość. Używają one słowa kluczowego **return** a wartość zwracaną przez nie możemy zapisać do zmiennej.

Może być to np.:

- Najróżniejsze obliczenia matematyczne (potęgowanie, pierwiastkowanie, etc.),
- Wczytywanie informacji z bazy danych,
- Wyszukiwanie danych w zbiorze.

# Dwa typy funkcji

Funkcje które nie zwracają wartości:

```
function sayHello($userName) {  
    echo("Hello " . $userName);  
}
```

Funkcja **nie** używa słowa kluczowego **return** czyli **nie** zwraca żadnej wartości.

```
$user = "Jacek";  
sayHello($user);
```

Funkcja nie zwraca wartości więc po prostu ją uruchamiamy.

Funkcje które zwracają wartość:

```
function sum($x, $y) {  
    $mySum = $x + $y;  
    return $mySum;  
}
```

Funkcja używa słowa kluczowego **return** czyli zwraca wartość.

```
$numSum = sum(32, 65);  
echo("Suma liczb = " . $numSum );
```

Jako że interesuje nas wynik tej funkcji to wartość przez nią zwracaną zapisujemy do zmiennej. Zmiennej tej możemy potem użyć w dalszej części naszego kodu.



Poprawne tworzenie funkcji  
i ponowne ich używanie  
jest bardzo ważne  
w programowaniu!

# Czas na zadania

- Przeróbcie ćwiczenia E.  
Ćwiczenie numer E1 i E2 zostanie rozwiązane przez wykładowcę.

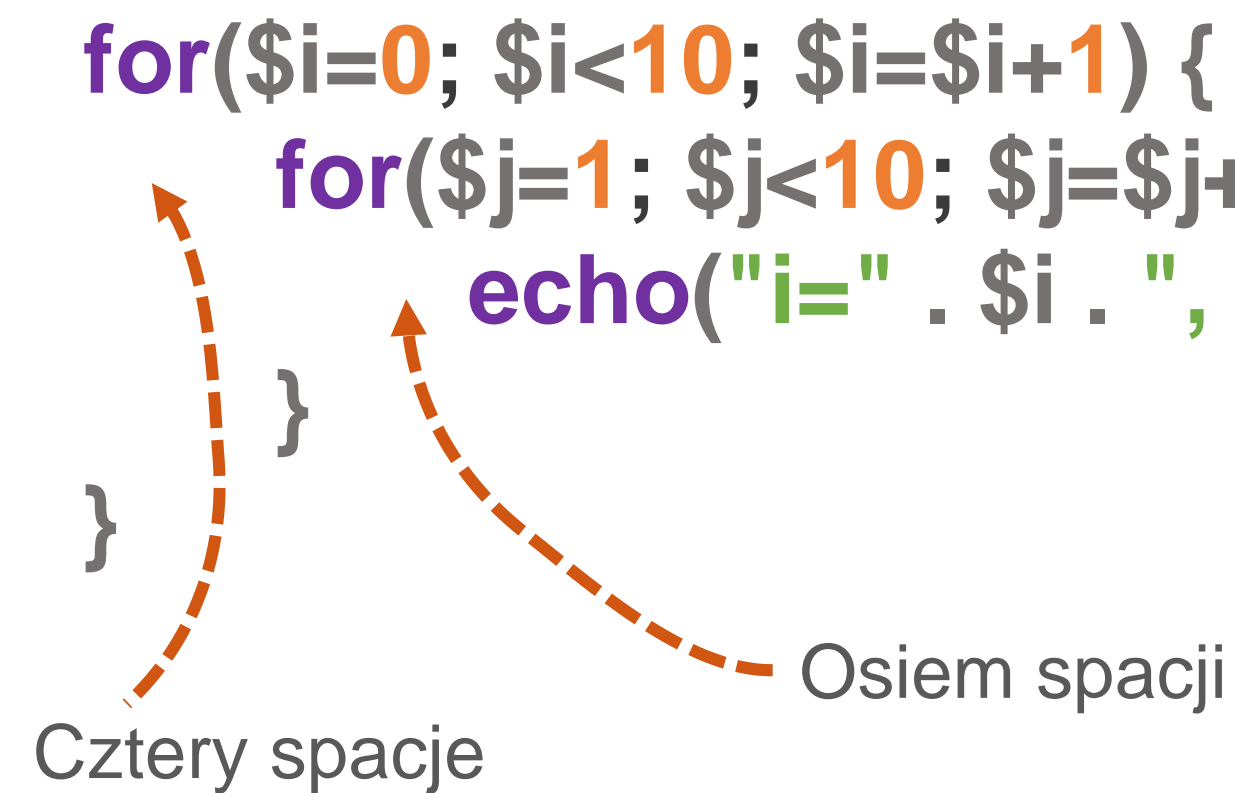


**Dobre praktyki**



# Używaj wcięć

- Bardzo ważna zasada – każdy blok kodu powinien dodawać dwie lub cztery spacje (nie używamy tabów!).
- Dzięki temu kod jest czytelniejszy i od razu widać, gdzie dany blok się kończy.



```
for($i=0; $i<10; $i=$i+1) {  
    for($j=1; $j<10; $j=$j+1) {  
        echo("i=" . $i . ", j=" . $j . "| ");  
    }  
}
```

Cztery spacje

Osiem spacji

# Używaj camelCase

- Nazwy zarówno zmiennych jak i funkcji zaczynamy od małej litery i używamy **camelCase**.

```
$myNumber = 76;
```

```
function writeMsg() {  
    echo("Hello world!");  
}  
writeMsg();
```

# Logicznie nazywajmy funkcje i zmienne

Wystrzegaj się zmiennych o nazwach \$test, \$bla, \$cosTam i innych tego typu.

```
$test = 30;  
$bla = 'Lorem';  
$cosTam = false;
```

# Praktyki ogólne

## Dziel kod!

Podział kodu na funkcje jest bardzo ważny – dzięki niemu kod jest bardziej czytelny i łatwiejszy w edytowaniu.

**Dziel kod na funkcje,  
ale nie pisz wielkich funkcji!**

## Don't repeat yourself – nie powtarzaj się!

Praktyka polegająca na nietworzeniu dwa razy tej samej funkcjonalności.

Unikamy więc kopiowania tych samych (lub bardzo podobnych) fragmentów kodu w wielu miejscach.

## Bez spacji w nazwach

Nie używaj spacji w nazwach plików i katalogów. Serwery bardzo często sobie z tym nie radzą, a problemy często objawiają się w dziwny sposób.

Dobłą praktyką jest nazywanie katalogów przy użyciu **camelCase** – ten nawyk przyda się później przy **przestrzeniach nazw (namespace)**.



Nie kopiuj kodu,  
którego nie rozumiesz!

# Najczęstsze błędy początkujących

# Najczęstsze błędy

## Łatwo widoczne (błędy interpretatora)

- brak średnika,
- zła nazwa funkcji,
- brak klamer, nawiasów, cudzysłowów zamykających blok.

Te błędy będą zazwyczaj wychwycone przez twoje IDE (środowisko programistyczne) i zostaną podkreślone jeszcze podczas pisania kodu.

## Mniej widoczne (błędy logiczne)

- użycie złej zmiennej,
- niezainicjowanie zmiennej,
- porównanie różnych typów danych.

Te błędy zazwyczaj spowodują że nasz program będzie działał ale wyniki będą inne od zakładanych (błędy logiki).



**Linki  
przydatne  
każdemu  
programiście**



# W3Schools

Znajdziecie tu referencje i podstawowe tutoriale dotyczące wszystkich języków związanych z programowaniem aplikacji internetowych:

- HTML + CSS,
- JavaScript i jQuery,
- PHP.

<http://www.w3schools.com>



# Dokumentacja PHP

Podstawowe miejsce, żeby dowiedzieć się,  
do czego służy funkcja.

Tłumaczy podstawy używania funkcji.

Zanim użyjemy nowej funkcji, warto tu zajrzeć!



<http://php.net/docs.php>

# Stackoverflow

Zawiera odpowiedzi na ogromną liczbę pytań.  
Bardzo przydatne zarówno początkującym,  
jak i zaawansowanym.

Zachęcamy do zaglądania – rozstrzał zagadnień  
jest tak duży, że każdy znajdzie coś dla siebie do  
poczytania!

<http://stackoverflow.com>

