

Podstawy PHP

Dzień II

v.1.3

Plan zajęć

- Funkcje
- Tablice
- Stringi czyli ciągi znaków
- Funkcje związane z datą

Funkcje

Funkcje

- Funkcja to wydzielony fragment kodu o zdefiniowanej nazwie, który może być używany w różnych momentach i miejscach programu.
- Funkcje można opisać jak rozwiązanie jakiegoś ogólnego problemu – przy tym dokładne informacje (zmienne) potrzebne do rozwiązania będą podane dopiero przy użyciu tej funkcji.
- Funkcję definiuje się raz, a później wywołuje się ją dowolną liczbę razy, w zależności od potrzeb i konieczności. Funkcja musi posiadać unikatową nazwę.

```
function nazwa(parametr1, parametr2, /* ... */,  
               parametrN) {  
    instrukcja1;  
    instrukcja2;  
    ...  
    instrukcjaN;  
  
    return zwracana_wartosc;  
}
```

Funkcje – definiowanie

- Nazwa funkcji może się składać z małych lub dużych liter, cyfr i znaków podkreślenia. Musi się zaczynać od litery lub znaku podkreślenia, ale nie musi zaczynać się od znaku `_` (w przeciwieństwie do zmiennych). W definicji nazwa funkcji jest poprzedzona słowem kluczowym **function**.
- Po nazwie funkcji następuje lista parametrów, które funkcja ma przyjmować. Parametry oddzielamy od siebie przecinkiem.
- Całą listę parametrów obejmujemy nawiasami okrągłymi. Jeżeli funkcja ma nie przyjmować parametrów, wtedy lista parametrów ma postać `()`.
- Instrukcja **return** zwraca wartość (wynik działania) funkcji i kończy jej działanie. Wszelkie instrukcje umieszczone za instrukcją **return** nie zostaną wykonane (martwy kod).

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 – wywoływanie

Zdefiniowaną funkcję możemy wywołać w dowolnym miejscu skryptu (nawet przed jej deklaracją – silnik PHP sam ją odnajdzie) przez podanie jej nazwy a następnie, w nawiasach okrągłych, jej parametry.

```
function countGross($net, $tax) {  
    $gross = $net + ($net * $tax);  
    return $gross;  
}  
  
$netPrice = 100;  
$taxInPoland = 0.23;  
  
$grossPrice = countGross($netPrice, $tax);  
  
echo("Cena brutto wynosi $grossPrice ");  
  
//bez użycia zmiennych  
echo('Cena brutto wynosi ');  
echo(countGross(100, 0.23) );
```


Funkcje - argumenty

- Funkcja może przyjmować argumenty. Są to zmienne które są potrzebne do prawidłowego działania funkcji, i są dostarczane podczas jej uruchomienia.
- Argumentów możemy przyjmować dowolną ilość.
- O argumentach musicie myśleć jak o zmiennych względem których będzie pracować wasza funkcja. Pracuje się na nich jak na każdej innej zmiennej, z tym wyjątkiem że do chwili użycia funkcji nie znamy ich dokładnej wartości.

```
function countGross($net, $tax) {  
    $gross = $net + ($net * $tax);  
    return $gross;  
}
```

Podczas pisania funkcji używamy ich jak każdej innej zmiennej.

Nasza funkcja **countGross** przyjmuje do siebie dwa argumenty: cenę netto i podatek

```
$netPrice = 100;  
$taxInPoland = 0.23;  
$grossPrice = countGross($netPrice, $tax);
```

Podczas wywołania funkcji musimy je podać

Funkcje – przekazywanie argumentów

- PHP udostępnia dwa sposoby przekazywania parametrów do funkcji – przez wartość i przez referencję.
- Parametr przekazany przez wartość jest podczas wywołania funkcji kopiowany do nowo utworzonej zmiennej lokalnej wewnątrz funkcji.
- Wszelkie zmiany wartości przekazanego parametru, jakie zostaną dokonane wewnątrz funkcji, nie będą obowiązywały poza nią – funkcja operuje na lokalnej kopii parametru.

```
function calculateTip($baseAmount) {  
    //10% napiwku  
    $tip = $baseAmount * 0.1;  
    $baseAmount += $tip;  
}
```

```
$myAmount = 50;  
calculateTip($myAmount);  
echo("razem do zapłaty: $myAmount ");
```

Podczas używania funkcji przekazujemy argument który jest kopiowany do **nowej** zmiennej o nazwie **\$baseAmount**

Funkcje – przekazywanie parametrów

- Drugim sposobem przekazania parametru jest użycie referencji. Referencje można opisać jako „alias” jakiejś zmiennej. Oznacza to że mamy jedną zmienną dostępną dla nas pod dwiema nazwami.
- Na liście parametrów funkcji parametr, który chcemy przekazać przez referencję, poprzedzamy znakiem **&**.
- Wszelkie zmiany wartości przekazanego parametru, jakie zostaną dokonane wewnątrz funkcji, będą obowiązywały także poza zasięgiem funkcji.

```
function calculateTip(&$baseAmount) {  
    //10% napiwku  
    $tip = $baseAmount * 0.1;  
    $baseAmount += $tip;  
}
```

Zmienne **\$baseAmount** i **\$myAmount** to jedna zmienna która teraz ma dwie nazwy.

```
$myAmount = 50;  
calculateTip($myAmount);  
echo("razem do zapłaty: $myAmount ");
```

Funkcje – domyślna wartość parametru

- Parametry funkcji mogą mieć zdefiniowane wartości domyślne.
- Wartość domyślną podajemy po nazwie parametru.
- Jeśli parametr ma domyślną wartość, nie musimy podawać jego wartości przy wywołaniu funkcji.
- Domyślna wartość parametru zostanie zignorowana, jeśli w wywołaniu funkcji podamy inną wartość.

```
function countGross($net, $tax = 23) {  
    $gross = $net + ($net * $tax);  
    return $gross;  
}
```

Argument **\$tax** ma wartość domyślną nastawioną na 23. Jeżeli wywołując tą funkcję go nie podamy zostanie do niego włożona wartość domyślna

```
$netPrice = 100;  
$grossPrice = countGross($netPrice);
```

Czas na zadania

- Przeróbcie ćwiczenia z katalogu „Funkcje”.
Pierwsze ćwiczenie zróbcie z wykładowcą.



Stringi czyli ciągi znaków

Operacje na stringach

W PHP zmienne typu **string** traktowane są tak samo jak tablice, co pozwala na dostęp do każdego znaku w stringu, tak jak do elementu tablicy.

```
$imie = 'Łukasz';  
echo($imie[1]); //u
```

Stringi możemy łączyć za pomocą operatora kropki (jest to tak zwana **konkatenacja** napisów).

```
$imie = 'Łukasz';  
$nazwisko = 'Pokrzywa';  
$imieNazwisko = $imie . ' ' . $nazwisko;  
echo($imieNazwisko); //Łukasz Pokrzywa
```

Operacje na stringach

Istnieje również specjalny operator:

`.=`

Jest on operatorem przypisania dla stringów z zachowaniem aktualnej wartości zmiennej.

```
$tekst = 'łatwo jest programować';  
$tekst .= ' w PHP';  
echo($tekst); //łatwo jest programować w PHP
```


Stringi – wycinanie fragmentu

Wycinanie fragmentu stringa z innego stringa możliwe jest przy użyciu funkcji:

substr(\$string, \$start, \$dlugosc)

- Pierwszym parametrem jest string, z którego będzie wycinany fragment,
- drugim – miejsce, z którego będzie rozpoczęte wycinanie (0 jeśli od pierwszego znaku, liczba ujemna – jeśli ma to być liczba znaków od końca),
- trzeci, opcjonalny parametr wskazuje liczbę znaków do wycięcia, jeśli go pominiemy, to zwrócony zostanie fragment od wskazanego znaku początkowego do końca stringa.

```
$tekst = 'laboratorium';
```

```
$foo = substr($tekst, 2, 5);  
echo($foo); //borat  
echo('<br>');
```

```
$foo = substr($tekst, -6, 3);  
echo($foo); //tor  
echo('<br>');
```

```
$foo = substr($tekst, 3);  
echo($foo); //oratorium
```

Stringi – rozbijanie i sklejanie

- **explode(\$separator, \$string)** – dzięki tej funkcji można rozbić string na kilka mniejszych fragmentów.
- Miejscami podziału będą wystąpienia znaku podanego w pierwszym parametrze funkcji.
- Funkcja zwraca tablicę, której elementami są kolejne fragmenty rozbitego stringa.

W odwrotną stronę działa funkcja **implode(\$klej, \$tablica)**. Skleja ona w jeden łańcuch elementy tablicy podanej w drugim parametrze, wstawiając pomiędzy nimi znak podany jako pierwszy parametr.

```
$godzina = '12:34:02';  
$tablicaGodzina = explode(':', $godzina);  
var_dump($tablicaGodzina);
```

```
$tablicaGodzina = array('12', '34', '02');  
$godzina = implode(':', $tablicaGodzina);  
echo($godzina);
```

Funkcje związane z data

Data i czas – date()

- Podstawową funkcją do pobierania daty i czasu jest **date()**.
- Domyślnie zwraca ona aktualną datę i czas z użyciem formatowania przekazanego jako pierwszy parametr.
- W drugim parametrze możemy przekazać timestamp – znacznik czasu.
- Pełną listę opcji znajdziesz w dokumentacji: <http://php.net/manual/pl/function.date.php>

//Przyjmujemy, że jest 10.03.2015 r., 14:09:36

```
echo(date('H:i:s')); //14:09:36
```

```
echo(date('m.d.y')); //03.10.15
```

```
echo(date("j, n, Y")); //28, 3, 2015
```

```
echo(date('h-i-s, j-m-y')); //02-09-36, 10-03-15
```

```
echo(date('F j, Y, g:i a')); //March 10, 2015, 2:09 pm
```

```
echo(date("D M j G:i:s T Y")); //Tue Mar 10
```

```
14:09:36 CET 2015
```

Data i czas – date(), time()

- Wspomnieliśmy, że **date()** zwraca aktualną datę. Jeśli chcemy dostać datę inną niż aktualną, należy do tej funkcji przekazać drugi parametr – **timestamp**.
- **timestamp** to liczba sekund, które upłynęły od 1.1.1970 00:00:00 (moment ten nazywany jest Epoch).
- Funkcja **time()** zwraca liczbę sekund, które upłynęły od Epoch do teraz.
- W ten sposób tworzymy datę względną – od aktualnej daty.

```
//liczba sekund od Epoch  
$aktualnyCzas = time();
```

```
//godzina ma 3600 sekund  
$zaGodzine = $aktualnyCzas + 3600;
```

```
//wyświetli datę 'za godzinę'  
echo(date("r", $zaGodzine));
```

```
// wyświetli datę '2 dni temu'  
$dwaDniTemu = $aktualnyCzas - (3600 * 24 * 2);  
echo(date("r", $dwaDniTemu));
```

Data i czas – strtotime()

- Innym sposobem na utworzenie daty względnej jest użycie funkcji **strtotime()**.
- Przekształca ona opisowe określenie daty na czas w formacie **timestamp**.

- Czas taki można przekazać później jako parametr funkcji **date()**, aby otrzymać datę w formacie przystępnym dla użytkownika.

Data i czas – strtotime()

//teraz

```
$teraz = strtotime("now");
```

//jutro

```
$jutro = strtotime("+1 day");
```

//za tydzień

```
$zaTydzien = strtotime("+1 week");
```

//za tydzień, dwa dni, 4 godziny i 2 sekundy

```
$kiedys = strtotime("+1 week 2 days 4 hours 2 seconds");
```

//następny czwartek

```
$nastepnyCzwartek = strtotime("next Thursday");
```

//poprzedni poniedziałek

```
$poprzedniPon = strtotime("last Monday");
```

```
echo(date('d.m.Y H:i:s', $jutro));
```

```
echo(date('d.m.Y H:i:s', $nastepnyCzwartek));
```

```
echo(date('d.m.Y H:i:s', $poprzedniPon));
```


Data i czas – konwersja do timestamp

- Jeżeli mamy poszczególne składniki daty (rok, miesiąc, dzień, godzina, itd...) i chcemy uzyskać reprezentację tej daty w formacie **timestamp**, to możemy użyć funkcji **mktime()**.
- Jako parametry przyjmuje ona odpowiednio godzinę, minutę, sekundę, miesiąc, dzień, rok i opcjonalnie informację o tym, czy jest to czas letni (0) czy zimowy (1).

```
$dzien = 10;  
$miesiac = 3;  
$rok = 2015;  
$godzina = 14;  
$minuta = 09;  
$sekunda = 36;
```

```
$date = mktime($godzina, $minuta, $sekunda,  
$miesiac, $dzien, $rok);
```

Data i czas – sprawdzanie poprawności

- Dzięki funkcji **checkdate()** możemy sprawdzić poprawność daty (bez godziny). Jako parametr przyjmuje ona odpowiednio miesiąc, dzień i rok.
- Funkcja ta sprawdza, czy rok ma wartości pomiędzy 1 a 32767, miesiąc między 1 a 12 a dzień od jeden do liczby zależnej od danego miesiąca. Uwzględniane są lata przestępne.
- Funkcja zwraca wartość **true**, jeżeli data jest poprawna, w przeciwnym wypadku – **false**.

```
$dzien = 10;  
$miesiac = 3;  
$rok = 2015;
```

```
var_dump(checkdate($miesiac, $dzien, $rok));  
var_dump(checkdate(2, 29, 2015));  
var_dump(checkdate(13, 21, 2015));
```

Data i czas – porównywanie dat

- Często trzeba porównywać daty i określić, która jest wcześniejsza lub późniejsza.
- Pamiętajmy, że aby dwie zmienne mogły być porównane, muszą być zmiennymi tego samego typu.
- W PHP nie ma typu danych dotyczących dat – najwygodniej porównywać je po przekonwertowaniu do formatu **timestamp**.
- Przy porównywaniu dat w formacie **timestamp** możemy używać operatorów **==**, **<**, **>**.

```
$data1 = '2015-03-09';  
$data2 = '2015-06-11';  
$data1Arr = explode('-', $data1);  
$data2Arr = explode('-', $data2);  
$data1Ts = mktime(0, 0, 0, $data1Arr[1],  
                  $data1Arr[2], $data1Arr[0]);  
$data2Ts = mktime(0, 0, 0, $data2Arr[1],  
                  $data2Arr[2], $data2Arr[0]);  
  
var_dump($data1Ts == $data2Ts);  
var_dump($data1Ts > $data2Ts);  
var_dump($data1Ts < $data2Ts);
```

Tablice

Tablice

Tablice to struktury danych przechowujące zbiór danych, zwykle jednego typu, do których możemy się odwoływać poprzez jedną wspólną nazwę tablicy. Każdy element tablicy jest zapisany pod unikalnym indeksem.

Tablice można tworzyć na trzy sposoby:

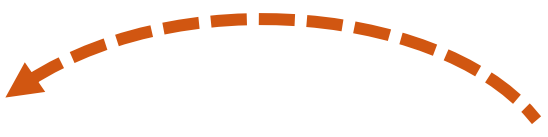
- przypisując na bieżąco wartości dla poszczególnych indeksów,
- za pomocą konstrukcji **array()**
- za pomocą konstrukcji **[]** (od PHP 5.4)

Pamiętaj że w informatyce tablice indeksujemy od 0!

```
$array[0] = 1;  
$array[1] = 2;  
$array[2] = 3;  
$array[3] = 4;  
$array[4] = 5;
```

```
$array2 = array(4, 5, 6, 7, 8);
```

```
$array3 = [7, 8, 9, 10];
```



Tablica o pięciu elementach będących liczbami całkowitymi.

Tablice asocjacyjne

W PHP można tworzyć następujące tablice:

- zwykłe (indeksowane kolejnymi nieujemnymi liczbami całkowitymi zaczynając od zera),
- asocjacyjne – indeksowane kluczem, którym może być również ciąg znaków.

```
$namesArray = array(  
    'imię'      => 'Jan',  
    'nazwisko'  => 'Kowalski',  
    'miasto'    => 'Warszawa',  
    'ulica'     => 'Hoża'  
);
```


Wyświetlanie elementów tablicy

- Możemy wyświetlać poszczególne elementy tablicy dzięki odwołaniu się oddzielnie do każdego z nich lub za pomocą pętli.
- Do ustalenia rozmiaru (liczby elementów) tablicy można użyć funkcji **count()**, która jako parametr przyjmuje tablicę, a zwraca liczbę całkowitą będącą rozmiarem tablicy.

```
$array = array(1, 2, 3, 4, 5);
```

```
echo($array[0]);
```

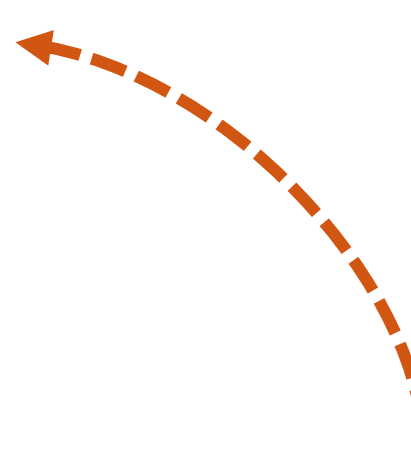
```
echo($array[1]);
```

```
echo($array[2]);
```

```
echo($array[3]);
```

```
echo($array[4]);
```

```
for($i = 0; $i < count($array); $i++) {  
    echo($array[$i]);  
}
```



Funkcja **count** zwraca nam ilość elementów w tablicy.

Wyświetlanie elementów tablicy

- Do pracy na elementach tablicy możemy również użyć pętli **foreach**.
- Jest to specjalna pętla w PHP która przechodzi po kolei po każdym elemencie w tablicy.
- Można jej użyć żeby w każdym przebiegu dostać kolejny element, albo tak żeby dostać zarówno element jak i klucz pod którym znajduje się dany element.

```
$namesArray = array(  
    'imię'      => 'Jan',  
    'nazwisko'  => 'Kowalski',  
    'miasto'    => 'Warszawa'  
);
```

W zmiennej `$name` będą kolejne elementy z tablicy `$namesArray`

```
foreach($namesArray as $name) {  
    echo("Imię: $name <br>");  
}
```

Dodatkowo możemy otrzymać jeszcze wartość klucza do zmiennej nazwanej `$key`

```
foreach($namesArray as $key => $name) {  
    echo("Imię pod kluczem $key to: $name <br>");  
}
```

Operacje na tablicach

+

- Tak jak w przypadku zmiennych innych typów na tablicach również można przeprowadzać operacje z użyciem poznanych już operatorów.
- Operator **+** służy do dodawania tablic.
Wynikiem jest:
 - tablica zawierająca wszystkie elementy z tablicy stojącej po lewej stronie operatora **+**;
 - oraz elementy z tablicy po prawej stronie operatora, które nie powtarzają się w tablicy z lewej strony operatora.

```
$dataArray = array(  
    'imie'      => 'Jan',  
    'nazwisko' => 'Kowalski',  
    'miasto'    => 'Warszawa'  
);
```

```
$imageArray = array(  
    'wzrost'    => '180',  
    'kolorOczu' => 'niebieskie'  
);
```

```
$lientData = $dataArray + $imageArray;
```

Operacje na tablicach

==, !=, <>

- Operator == sprawdza, czy tablice są równe. Dwie tablice są równe, gdy mają **taką samą liczbę elementów** oraz **te same wartości** (tablice klasyczne) lub **pary klucz => wartość** (tablice asocjacyjne). Kolejność elementów nie jest istotna.
Typy elementów nie są sprawdzane.

- Operatory != oraz <> sprawdzają, czy tablice są różne. Dwie tablice są różne, jeśli nie jest spełniony przynajmniej jeden z warunków równości tablic.

```
$client1 = array(  
    'imie'      => 'Anna',  
    'miasto'    => 'Warszawa'  
);  
  
$client2 = array(  
    'miasto'    => 'Warszawa',  
    'imie'      => 'Anna'  
);  
  
var_dump($client1 == $client2);
```

Operacje na tablicach

===, !==

- Operator `===` sprawdza, czy tablice są identyczne. Dwie tablice są identyczne, gdy mają **taką samą liczbę elementów** oraz te **same wartości** (tablice klasyczne) lub **pary klucz => wartość** (tablice asocjacyjne). Kolejność elementów jest istotna. **Typy elementów muszą się zgadzać.**

- Operator `!==` sprawdza, czy tablice nie są identyczne. Dwie tablice nie są identyczne, jeśli nie jest spełniony przynajmniej jeden z warunków identyczności tablic.

```
$client1 = array(  
    'imie' => 'Anna',  
    'miasto' => 'Warszawa',  
    'wiek' => '28'  
);  
  
$client2 = array(  
    'imie' => 'Anna',  
    'miasto' => 'Warszawa',  
    'wiek' => 28  
);  
  
var_dump($client1 === $client2);
```

Sortowanie tablic – sort(), rsort()

- PHP ma zestaw funkcji pozwalających na sortowanie tablic.
 - Funkcję wybieramy w zależności od rodzaju tablicy, którą chcemy posortować (klasyczna lub asocjacyjna), oraz kolejności, w jakiej mają być posortowane elementy.
 - Wszystkie funkcje sortujące operują bezpośrednio na tablicy przekazanej w parametrze.
- Funkcje sortujące nie zwracają żadnej wartości.
 - Funkcja **sort()** sortuje tablice (klasyczne), układa elementy od najmniejszego do największego.
 - Funkcja **rsort()** układa elementy od największego do najmniejszego.

Sortowanie tablic – asort(), arsort()

- Funkcja **asort()** sortuje tablice asocjacyjne. Układa ona elementy od najmniejszego do największego.
- Funkcja **arsort()** sortuje tablice asocjacyjne w porządku odwrotnym.
- Obydwie funkcje sortują tablice tak, że klucze zachowują przypisanie do odpowiednich wartości.
- Do tablic asocjacyjnych powinniśmy używać funkcji dla nich przeznaczonych. Użycie funkcji **sort()** lub **rsort()** może zmienić przypisanie klucza do wartości.

```
$capitals = array(  
    'Kanada'    => 'Ottawa',  
    'Niemcy'    => 'Berlin',  
    'Austria'   => 'Wiedeń',  
    'Japonia'   => 'Tokio'  
);
```

```
asort($capitals );  
var_dump($capitals);
```

```
arsort($capitals);  
var_dump($capitals);
```

Sortowanie tablic – ksort()

- Funkcja **ksort()** sortuje tablice asocjacyjne według kluczy.
- Nazwa jest skróconym zapisem keySort.


```
$capitals = array(  
    'Kanada'    => 'Ottawa',  
    'Niemcy'    => 'Berlin',  
    'Austria'   => 'Wiedeń',  
    'Japonia'   => 'Tokio'  
);  
  
ksort($capitals);  
var_dump($capitals);
```


Tablice wielowymiarowe

- Dotychczas zajmowaliśmy się tylko tablicami jednowymiarowymi (postać wektora).
- Można zdefiniować tablice wielowymiarowe (w praktyce rzadko stosuje się tablice większe niż dwuwymiarowe).
- Tablice wielowymiarowe pozwalają na swego rodzaju zagnieżdżanie tablicy w tablicy.

```
$tab1 = array(1, 2, 3);  
$tab2 = array(11.5, 45.3, 0.43);  
$array2D = array($tab1, $tab2);
```

Zmienna **\$tab** będzie w sobie miała kolejne tablice



```
foreach($array2D as $tab) {  
  
    foreach($tab as $dataFromTab) {  
        echo($dataFromTab);  
    }  
}
```

Czas na zadania

- Przeróbcie ćwiczenia z katalogu „Tablice”.
Pierwsze ćwiczenie zróbcie z wykładowcą.

KONIEC