SAREscript

Dokumentacja funkcjonalna

wersja dokumentu 1.0

Tomasz Kusy

t.kusy@sare.pl

I. Wstęp

Niniejszy dokument jest opisem języka skryptowego SAREscript w wersji 1.0. Zawiera on informacje o składni języka, instrukcjach, funkcjach i innych elementach niezbędnych do praktycznego wykorzystania jego możliwości.

Ta wersja dokumentu (1.0) jest wersją przeznaczoną dla partnerów SARE i innych osób planujących używanie SAREscript.

Dokument zawiera podstawową dokumentację użytkową SAREscript, umożliwiającą samodzielne tworzenie skryptów SAREscript, oraz spis dostępnych funkcji SAREscript. Opisy funkcji zostały uzupełnione o praktyczne przykłady ich stosowania.

SARE sp. z o.o. zastrzega sobie prawo do swobodnej zmiany, modyfikacji, usunięcia dowolnej części tej dokumentacji, jak i składni bądź działania opisanych w tej dokumentacji funkcji, instrukcji i innych elementów, jeżeli będzie tego wymagało wyeliminowanie potencjalnych błędów, rozwój systemu bądź inne ważne przyczyny. Informacje o zmianach będą dostępne na stronie www.sare.pl oraz w newsletterze SARE.

II. Podstawowe informacje o SAREscript

1. Co to jest SAREscript

SAREscript jest językiem, który powstał na potrzeby systemu SARE. Ma on za zadanie umożliwić osobom korzystającym z systemu SARE pełniejszą kontrolę nad treścią wysyłanych wiadomości e-mail. Zanim powstał SAREscript, SARE umożliwiało jedynie prostą personalizację wiadomości, poprzez podstawianie w wybrane miejsca przechowywanych w bazie cech danego odbiorcy. SAREscript umożliwia manipulację danymi odbiorcy (np. podczas wysyłki) oraz pozwala na przetwarzanie innych danych dostępnych w systemie. Na danych można wykonywać operacje czy to za pomocą podstawowych operatorów (np. łączenie, dodawanie, odejmowanie, mnożenie, dzielenie), jak i dostępnych funkcji (np. zaokrąglanie liczb, fleksja imion, słowne przedstawianie liczb, tworzenie newsletterów, planowanie wysyłek). Bardzo ważną cechą SAREscriptu jest możliwość tworzenia konstrukcji warunkowych (if, elseif, else). Istnieje możliwość tworzenia pętli (for). Dostępna jest możliwość definiowania i używania własnych zmiennych, w tym tablic. Obecnie SAREscript zawiera kilkadziesiąt funkcji, jednak będzie stale rozbudowywany.

2. Podobieństwa do innych języków

SAREscript jest w dużej mierze wzorowany na jednym z najpopularniejszych języków skryptowych - PHP. Osoba znająca język PHP nie będzie miała problemów z jego zrozumieniem i używaniem. W niniejszym dokumencie można będzie znaleźć wiele odwołań do PHP. Nazwy funkcji SAREscript mające swe odpowiedniki (jeżeli chodzi o działanie) w PHP bardzo często mają taką samą nazwę. To wszystko powinno umożliwić łatwe korzystanie z języka SAREscript programistom przyzwyczajonym do korzystania z PHP, a jednocześnie pozwalać na tworzenie nawet zaawansowanych skryptów osobom nie mającym nic wspólnego z programowaniem.

Nawiązanie do PHP nie jest wymuszone, a celowe i zamierzone. Nie ma bowiem sensu (aczkolwiek przy projektowaniu SAREscript nic nie stało temu na przeszkodzie) tworzenia własnej, unikalnej składni języka. Jednak na przykład składnia instrukcji pętli *for* jest bliższa znanej np. z Basica.

3. Używanie SAREscriptu

Używanie SAREscript w systemie SARE możliwe jest poprzez utworzenie programu, który zostanie zinterpretowany i wykonany przez system. SAREscript używany jest m.in. podczas wysyłki, przy wyświetlaniu mapy kliknięć, personalizowanym podglądzie www czy jako filtr zaawansowany. Program może zostać wywołany samodzielnie bądź osadzony w treści wiadomości, która ma być wysłana. System

SARE sam sprawdza, czy w treści wiadomości używany jest SAREscript. Jeżeli jest użyty podczas wysyłki, dla każdego odbiorcy wiadomości wykonywany jest skrypt z użyciem danych (cech) przypisanych do adresu, na który prowadzona jest wysyłka. W jednej wiadomości e-mail może być osadzonych więcej niż jeden blok ze skryptem. Taki pojedynczy blok nazywamy **scriptspotem**.

4. Scriptspot

SAREscript używany jest poprzez stworzenie odpowiedniego scriptspotu. Scriptspot rozpoczynamy przez umieszczenie na jego początku: `<!--sare`

Scriptspot zamykamy poprzez: `sare-->`

Znacznik zamykający musi zostać poprzedzony „białym znakiem” (nowa linia, spacja, etc.)

Przykładowy scriptspot:

```
<!--sare print("To jest scriptspot"); sare-->
```

Scriptspot może być zawarty w jednej (jak powyżej) bądź wielu liniach:

```
<!--sare
print("To jest scriptspot");
sare-->
```

W obu wersjach odbiorca zobaczy w swojej wiadomości słowa:

```
To jest scriptspot
```

Niewłaściwy scriptspot:

```
<!--sare print("To jest scriptspot");sare-->
```

Brak jest białego znaku pomiędzy średnikiem a zamknięciem scriptspota (`;sare-->` zamiast `; sare-->`)

Wewnątrz scriptspota nie powinien znaleźć się ciąg znaków `sare-->` . Może on znaleźć się jedynie na jego końcu.

5. Osadzanie scriptspota w wiadomości

Scriptspot może zostać osadzony w kodzie wiadomości tak jak inne znaczniki HTML, w dowolnej części kodu. Przykład:

```
<HTML>
 <HEAD>
  <TITLE>Newsletter</TITLE>
 </HEAD>
 <BODY>
Zachęcamy <!--sare print("Pana"); sare--> do odwiedzenia naszej strony <a
href='http://www.naszastrona.pl/'>www.naszastrona.pl</a>
```

```
 </BODY>
</HTML>
```

Można jednocześnie w kodzie umieścić więcej niż jeden scriptspot:

```
Zachęcamy <!--sare print("Pana"); sare--> do <!--sare print("ponownego"); sare-->
odwiedzenia naszej strony <a href='http://www.naszastrona.pl/'>www.naszastrona.pl</a>
```

UWAGA! Nie powinno się umieszczać scriptspota pomiędzy znakami otwarcia i zamknięcia < > znacznika HTML. **Błędne** jest takie osadzenie:

```
Zachęcamy Pana do odwiedzenia naszej strony <a href='http://www.naszastrona.pl/
<!--sare print("panowie"); sare-->'>www.naszastrona.pl</a>
```

Nie wolno również przecinać znaczników (np. znacznik otwarcia generowany przez SAREscript, zaś zamknięcie statyczne). Również **błędne** jest takie osadzenie:

```
Zachęcamy Pana do odwiedzenia naszej strony <!--sare print("<a
href='http://www.naszastrona.pl/panowie'>"); sare-->www.naszastrona.pl</a>
```

Prawidłowo powinno być np. tak:

```
Zachęcamy do odwiedzenia naszej strony <!--sare
print("<a href='http://www.naszastrona.pl/panowie'>www.naszastrona.pl</a>"); sare—>
```

6. Zależności pomiędzy scriptspotami

Używanie SAREscript w postaci scriptspota osadzonego w treści wiadomości jest jego najbardziej praktycznym zastosowaniem. Możliwe jest jednak zastosowanie SAREscript w innym kontekście (np. jako filtr zaawansowany). Poniżej miejsca (lub kontekst) w systemie, gdzie wykonywany jest SAREscript:

1. Mailing > edytor – podgląd wiadomości
2. Mailing > edytor – podgląd na e-mail
3. Mailing > wysyłanie – mail testowy
4. Mailing > wysyłanie – testowanie filtrów
5. Mailing > wysyłanie – wysyłka właściwa
6. SAREscript > uruchom – wykonanie i debugowanie
7. Raporty > mapa kliknięć
8. Podgląd www wiadomości
9. Scenariusze subskrypcji – wyświetlanie i wysyłanie obiektów
10. Interfejs SOAP – wywołanie metody Execute

Użycie SAREscript w powyżej przywołanych miejscach różni się od siebie w sposób istotny. Zdarza się, że skrypty przetwarzane są jeden po drugim, a kolejność ich wykonania ma zasadnicze znaczenie dla

poprawności procesu. Dlatego konieczne jest zrozumienie, jak wygląda proces uruchamiania scriptspotów. Szczegółowy opis znajduje się w **dodatku B**.

III. Składnia SAREscript

1. Podstawy

SAREscript jest językiem *case sensitive,* tzn. gdzie ważna jest wielkość liter. Komendy, funkcje i zmienne wymagają ich używania zgodnie z tym, jak zostały zadeklarowane. Większość funkcji jest pisana małymi literami. Na przykład istnieje funkcja `print`, ale nie ma `Print`, `PRINT` czy `PrInT` (gdyby istniały, to mogłyby to być zupełnie inne funkcje).

Wszystkie instrukcje, podobnie jak w wielu innych językach programowania, muszą być zakończone znakiem średnika **;** na końcu każdego wyrażenia (linii). Brak takiego zakończenia nie zawsze spowoduje wystąpienie błędu, jednak tego skutki pominięcia znaku **;** mogą być nieprzewidywalne.

2. Typy danych

Obecnie SAREscript używa następujących typów danych: NUMERIC (liczby), STRING (łańcuchy znaków), ARRAY (tablica), BOOL (logiczna prawda/fałsz) oraz specjalnego typu NULL. Liczby zmiennoprzecinkowe zapisujemy, oddzielając część dziesiętną od całkowitej za pomocą kropki (np. 0.5-2.123 1.0).

3. Łańcuchy znaków

Łańcuchy (string) możemy definiować za pomocą cudzysłowów **"** bądź apostrofów **'** . Dostępny jest również, znany z PHP zapis *heredoc*. Łańcuchem zostanie wszystko, co znajdzie się pomiędzy cudzysłowem (bądź apostrofem) otwierającym a zamykającym. Przykłady: `"To jest łańcuch" 'To też jest łańcuch'`.

W przypadku użycia cudzysłowu, możemy korzystać w nim ze znaków specjalnych przez użycie znaku ucieczki **** (backslash). Przykładowo:

```
print("To jest pierwsza linia.\nTo jest druga linia.");
```

zostanie zamienione na:

```
To jest pierwsza linia.
To jest druga linia.
```

natomiast

```
print('To jest pierwsza linia.\nTo jest druga linia');
```

zostanie wyświetlone tak jak wpisano:

```
To jest pierwsza linia.\nTo jest druga linia.
```

Innym przykładem jest użycie znaku ucieczki **** celem zamaskowania znaku **"** wewnątrz łańcucha. Przykładowo takie użycie stringa zakończy się błędem:

```
print("<img src="http://jakisadresobrazka.gif">");
```
Aby tego uniknąć, można zapisać:
```
print("<img src=\"http://jakisadresobrazka.gif\">");
```

Zapis ***heredoc*** przydatny jest przy definiowaniu dłuższych łańcuchów, np. fragmentów HTML. Użycie tego zapisu następuje poprzez zapisanie operatora **<<<** za którym bezpośrednio znajduje się *identyfikator* i znak nowej linii. Łańcuch zaczyna się w następnej linii i kończy, kiedy wystąpi ten sam *identyfikator*, jak użyty na początku. *Identyfikator* zamykający musi zacząć się od pierwszej kolumny kodu i nie może w tej samej linii być żadnych innych znaków, jedynie opcjonalnie dopuszczony jest znak **;**. *Identyfikator* może zawierać tylko litery, cyfry oraz znak _ (podkreślenie), ale musi zaczynać się od litery.

Przykładowo:
```
print(<<<EOT
  <a href="http://www.sare.pl/">
    <img src="http://www.sare.pl/logo.gif">
  </a>
EOT
);
```

W przeciwieństwie do PHP, znak **$** występujący w łańcuchach jest traktowany jako zwykły znak, a nie jako początek zmiennej. To znaczy, że podczas parsowania łańcuchów nie zamienia się występujących w łańcuchach odwołań do zmiennych na ich wartości.

SAREscript pracuje w kodowaniu UTF-8 i w takim kodowaniu przechowuje wszystkie łańcuchy.

4. Zmienne

SAREscript umożliwia używanie dowolnej liczby zmiennych. Nazwa zmiennej rozpoczyna się od znaku dolara $, za nim litery (bez znaków narodowych) i opcjonalnie kolejnych znaków spośród liter, cyfr lub znaku podkreślenia _. Poprawną nazwą zmiennej będzie np.: `$moja_zmienna2` , ale już nie `$2moja_zmienna`.

Dostępne są również zmienne tablicowe (tablice). Tablice pozwalają na dostęp do wartości w nich przechowywanych z wykorzystaniem *klucza*. Klucz może być typu NUMERIC lub STRING.. Tablice są tworzone jako wynik działania funkcji (np. array()) bądź poprzez przypisanie elementowi tablicy (wskazanemu przez klucz) dowolnej wartości (w tym innej tablicy):
```
$zmienna[klucz] = wartość
```
Przykładowo wszystkie poniższe deklaracje są poprawne i zmienna $liczby jest zmienną tablicową:
```
$liczby = array("zero", "jedynka");
$liczby[1] = "jedynka";
$liczby["jedynka"] = 1;
```

Zastrzeżonym kluczem tablicy jest słowo "SYSTEM". Tablica ta zawiera informacje o systemie i dlatego ten

klucz nie może być używany.

Jeżeli nie zaznaczono inaczej, większość funkcji, które zwracają w wyniku działania tablice, indeksuje te tablice od zera (pierwszy element tablicy ma indeks 0).

5. Operatory i wyrażenia

Obecnie SAREscript obsługuje następujące operatory:

a) arytmetyczne

Nazwa	Operator	Przykład	Wynik operacji
negacja	-	-$a	Zanegowanie zmiennej $a. Równoważne ($a * -1)
dodawanie	+	$a + $b	Suma $a i $b
odejmowanie	-	$a - $b	Różnica $a i $b
mnożenie	*	$a * $b	Iloczyn $a i $b
dzielenie	/	$a / $b	Iloraz $a i $b

b) przypisania

Nazwa	Operator	Przykład	Wynik operacji
przypisanie	=	$a = $b	Zmienna $a przyjmie wartość $b

c) inkrementacji/dekrementacji

Nazwa	Operator	Przykład	Wynik operacji
inkrementacja	++	$a++	Zmienna $a zwiększy swoją wartość o 1
dekrementacja	--	$a--	Zmienna $a zmniejszy swoją wartość o 1

d) porównania

Nazwa	Operator	Przykład	Wynik operacji
równe	==	$a == $b	Prawda jeżeli $a jest równe $b
różne	!=	$a != $b	Prawda jeżeli $a jest różne od $b
równe dokładnie	===	$a === $b	Prawda jeżeli $a jest równe $b i są tego samego typu
różne dokładnie	!==	$a !== $b	Prawda jeżeli $a jest różne od $b lub są innych typów
większe	>	$a > $b	Prawda jeżeli $a jest większe od $b
mniejsze	<	$a < $b	Prawda jeżeli $a jest mniejsze od $b

większe równe	**>=**	$a **>=** $b	Prawda jeżeli $a jest równe lub większe od $b
mniejsze równe	**<=**	$a **<=** $b	Prawda jeżeli $a jest równe lub mniejsze od $b

e) logiczne

Nazwa	**Operator**	**Przykład**	**Wynik operacji**
lub	**or**	$a **or** $b	Prawda jeżeli $a jest prawdą lub $b jest prawdą
i	**and**	$a **and** $b	Prawda jeżeli $a jest prawdą i $b jest prawdą

f) łańcuchowe

Nazwa	**Operator**	**Przykład**	**Wynik operacji**
łączenia	**.**	$a **.** $b	Połączone łańcuchy $a i $b

Priorytety operatorów (od góry najwyższe):

++ --
negacja
*** /**
+ - .
< > >= <=
== != === !==
=
or
and

6. Struktury kontrolne

Struktury kontrolne są jednym z najważniejszych elementów SAREscript. Dzięki nim możliwe jest warunkowe wykonywanie określonych elementów kodu. Podstawową strukturą, podobnie jak w innych językach programowania, jest **`if`**.

```
if (wyrażenie) {
      kod_do_wykonania
}
```

Jeżeli *`wyrażenie`* jest prawdą, zostanie wykonany kod zawarty pomiędzy nawiasami. Przykład:

```
if ($plec == "M") {
      print("Szanowny Panie");
}
```

Dodatkowe możliwości stwarza zastosowanie **`else`**.

```
if (wyrażenie) {
      kod_do_wykonania
} else {
      alternatywny_kod_do_wykonania
}
```

Jeżeli *`wyrażenie`* jest prawdą, zostanie wykonany *`kod_do_wykonania`*, natomiast jeżeli będzie fałszem, wykonany zostanie alternatywny *`kod_do_wykonania`*. Przykład:

```
if ($plec == "M") {
      print("Szanowny Panie");
} else {
      print("Szanowna Pani");
}
```

SAREscript pozwala również na stosowanie **`elseif`**, które jest połączeniem **`else`** i **`if`**

```
if (wyrażenie1) {
      kod_do_wykonania1
} elseif (wyrażenie2) {
      kod_do_wykonania2
} elseif (wyrażenie3) {
      kod_do_wykonania3
} else {
      kod_do_wykonania4
}
```

Jeżeli *`wyrażenie1`* jest prawdą, zostanie wykonany *`kod_do_wykonania`*, natomiast jeżeli będzie fałszem, sprawdzone zostanie *`wyrażenie2.`* Jeżeli to wyrażenie również będzie fałszem, sprawdzane będą kolejne wyrażenia przy *`elseif`* aż do skutku. Jeżeli żadne z wyrażeń nie będzie prawdą, to wykonany zostanie blok *`else`* (o ile został zdefiniowany). Przykład:

```
if ($wiek < 21) {
      print("Cześć");
} elseif ($wiek < 50) {
      print("Witam");
} else {
      print("Dzień dobry");
}
```

Struktury mogą być zagnieżdżone.

Jeżeli jako wyrażenie dla if występują wyrażenia wymagające wcześniejszego wykonania (obliczenia), to wszystkie zostaną wykonane, niezależnie od tego, czy są konieczne. Np.:

```
if (wyrażenie1 or wyrażenie2)
```

W przypadku kiedy *wyrażenie1* zwróci *false*, to i tak zostanie wykonane *wyrażenie2* (chociaż nie ma już wpływu na wynik całości sprawdzenia, które i tak będzie *false*). Jest to inne działanie niż np. w PHP.

7. Pętle

Pętla FOR pozwala na wywołanie określonego fragmentu skryptu więcej niż jeden raz. Składnia wygląda następująco:

```
for ($zmienna = start to stop) {
      kod do wykonania
}
```

Start to początkowa wartość zmiennej *$zmienna*, zaś *stop* to wartość końcowa. Jeżeli wartość *start* będzie mniejsza od *stop*, podczas każdego kroku wartość zmiennej jest większa o jeden do czasu, kiedy wartość zmiennej *$zmienna* będzie równa (bądź większa) wartości zdefiniowanej jako *stop*. W przypadku kiedy na początku wartość *start* jest większa od *stop*, zmienna *$zmienna* będzie w każdym kroku zmniejszana o jeden, aż jej wartość będzie równa bądź mniejsza od *stop*. Jeżeli *start* i *stop* będą sobie równe, kod zostanie wykonany dokładnie jeden raz.

Parametr *$zmienna* nie może być zmienną tablicową.

Przykład:

```
for ($licznik = 1 to 5) {
      print($licznik." razy ".$licznik." równa się ".($licznik*$licznik)."\n");
}
```

Wynikiem działania będzie:

```
1 razy 1 równa się 1
2 razy 2 równa się 4
3 razy 3 równa się 9
4 razy 4 równa się 16
5 razy 5 równa się 25
```

Przykład:

```
for ($licznik = 5 to 1) {
      print($licznik." razy ".$licznik." równa się ".($licznik*$licznik)."\n");
}
```

Wynikiem działania będzie:

```
5 razy 5 równa się 25
4 razy 4 równa się 16
3 razy 3 równa się 9
2 razy 2 równa się 4
1 razy 1 równa się 1
```

8. Rejestry

SAREscript posiada możliwość zdefiniowania zmiennych globalnych dla systemu zwanych rejestrami. Rejestry są polami w bazie, do których można odwoływać się poprzez ich nazwę. Nazwa rejestru może mieć maksymalnie 255 znaków. W jednym rejestrze można przechowywać łańcuchy znaków o maksymalnej długości do 1023 znaków lub 16777215 (16 MB) w przypadku rejestrów typu BIG.

9. Funkcje

Funkcje pozwalają na przekazanie im parametru. Zwracają wynik swojego działania. Obecnie SAREscript oferuje funkcje, które są specyficzne dla systemu e-mailingowego, jak również odpowiedniki funkcji PHP. Oprócz podstawowych funkcji (np. print()) są również zaawansowane funkcje, które operują na dużej liczbie danych (np. pl_imiewolacz()). Zwiększanie liczby dostępnych funkcji będzie głównym kierunkiem rozbudowy SAREscriptu. Szczegółowy opis funkcji znajduje się w dodatku A.

9.1. Funkcje przetwarzające teksty

lastchar – zwraca ostatni znak łańcucha

str_split – dzieli łańcuch na części

str_repeat – powtarza łańcuch określoną liczbę razy

strtolower – zamienia wielkie litery na małe

substr – zwraca określony fragment łańcucha

substr_count – sprawdza liczbę wystąpień jednego łańcucha w drugim

trim – usuwa białe znaki z początku i końca łańcucha

9.2. Funkcje matematyczne i operujące na zmiennych

array_search – wyszukuje w tablicy wartość

count – zwraca liczbę elementów tablicy

empty – sprawdza, czy argument jest „pusty”

explode – zwraca tablicę zawierającą podzielony łańcuch

in_array – sprawdza, czy w tablicy znajduje się określona wartość

intval – zamienia argument na liczbę całkowitą

not – neguje argument

ok – zawsze zwraca true

rand – zwraca losową liczbę z określonego przedziału

9.3. Funkcje czasu i daty

date – zwraca odpowiednio sformatowaną datę

time – zwraca aktualny uniksowy znacznik czasu

9.4. Funkcje wpływające na zachowanie się SARE

system_setfrom – ustawia pole From podczas wysyłki newslettera

system_setskipmessage – pomija wysyłkę newslettera na bieżący adres

system_setsubject – ustawia pole Subject podczas wysyłki newslettera

9.5. Funkcje operujące na danych z bazy i zewnętrznych

bigregistry_get – zwraca zawartość rejestru typu BIG

bigregistry_set – zapisuje wartość do rejestru typu BIG

bigregistry_unset – usuwa rejestr typu BIG

get_url – pobiera treść znajdującą się pod wskazanym adresem URL

get_val – zwraca dane skojarzone z bieżącym adresem e-mail

registry_get – zwraca zawartość rejestru

registry_set – zapisuje wartość do rejestru

registry_unset – usuwa rejestr

set_val – zapisuje do bazy dane skojarzone z bieżącym adresem e-mail

9.6. Funkcje operujące na danych statystycznych (raporty)

campaign_clicked – zwraca liczbę klikniętych wiadomości podczas wskazanej kampanii dla bieżącego adresu e-mail

campaign_ctr – zwraca wskaźnik CTR dla wskazanej kampanii, dla bieżącego adresu e-mail

campaign_opened – zwraca liczbę otwartych wiadomości podczas wskazanej kampanii dla bieżącego adresu e-mail

campaign_openrate – zwraca wskaźnik Openrate dla wskazanej kampanii, dla bieżącego adresu e-mail

campaign_sent – zwraca liczbę maili, jaką wysłano na bieżący adres e-mail podczas wskazanej kampanii

campaign_total_clicked – zwraca łączną liczbę kliknięć podczas wskazanej kampanii

campaign_total_ctr – zwraca wskaźnik CTR dla wskazanej kampanii

campaign_total_opened – zwraca wskaźnik Openrate dla wskazanej kampanii

campaign_total_openrate – zwraca liczbę otwartych wiadomości podczas wskazanej kampanii

campaign_total_sent – zwraca liczbę maili wysłaną podczas wskazanej kampanii

mailing_bounced – informuje, czy we wskazanej wysyłce dla bieżącego adresu e-mail zanotowano zwrot

mailing_clicked – informuje, czy we wskazanej wysyłce dla bieżącego adresu e-mail zanotowano kliknięcie

mailing_clicked_link – informuje, czy we wskazanej wysyłce dla wskazanego linka, dla bieżącego adresu

e-mail zanotowano kliknięcie

mailing_opened – informuje, czy we wskazanej wysyłce dla bieżącego adresu e-mail zanotowano otwarcie maila

mailing_planned_count – zwraca liczbę zaplanowanych wysyłek dla bieżącego adresu e-mail

mailing_sent – informuje, czy podczas wskazanej wysyłki został wysłany e-mail dla bieżącego adresu e-mail

mailing_sent_count – zwraca liczbę wysłanych e-maili na bieżący adres e-mail w określonym czasie

mailing_total_clicked – zwraca liczbę kliknięty wiadomości e-mail podczas wskazanej wysyłki

mailing_total_ctr – zwraca wskaźnik CTR dla wskazanej wysyłki

mailing_total_opened – zwraca liczbę otwartych wiadomości e-mail podczas wskazanej wysyłki

mailing_total_openrate – zwraca wskaźnik openrate dla wskazanej wysyłki

mailing_total_sent – zwraca liczbę wysłanych wiadomości e-mail podczas wskazanej wysyłki

sms_received – informuje, czy podczas wskazanej wysyłki SMS, dla bieżącego adresu e-mail, zanotowano odebranie SMS-a

sms_sent – informuje, czy podczas wskazanej wysyłki SMS, dla bieżącego adresu e-mail, wysłano SMS-a

9.7. Funkcje wyjścia

barcode_ean13 – osadza w treści wiadomości znacznik HTML odpowiedzialny za wyświetlenie kodu kreskowego EAN13

print – osadza w treści wiadomości (lub wyświetla jako wynik działania) łańcuch wskazany jako argument

print_r – osadza w treści wiadomości (lub wyświetla jako wynik działania) wartość przekazanej zmiennej w czytelnej postaci

9.8. Funkcje związane z newsletterem

mailing_info_get – zwraca tablicę z informacjami o wskazanej wysyłce

mailing_send – przeprowadza wysyłkę mailingu

mailing_send_test – przeprowadza testową wysyłkę mailingu

newsletter_create – tworzy w systemie nową kreację newslettera

9.9. Funkcje specjalne

pl_imiewolacz – zwraca formę wołaczową przekazanego imienia

pl_kody – oblicza odległość pomiędzy dwoma miejscowościami na podstawie kodów pocztowych

pl_slownie – przetwarza liczbę na zapis słowny

sare_groups – zwraca tablicę zawierającą numery grup, do których należy bieżący adres e-mail

sare_in_group – zwraca informację, czy bieżący adres e-mail należy do wskazanej grupy

Dodatek A: Opis funkcji

Przyjęto następująca konwencję opisu funkcji. Przed nazwą funkcji podany jest rodzaj zwracanej wartości przez funkcję. Jeżeli funkcja nie zwraca żadnych danych, oznaczono to słowem *void*. Jeżeli wynik funkcji może zwrócić wynik różnego typu, oznaczono to słowem *mixed*. W nawiasie umieszczono rodzaj wymaganych parametrów. Parametry opcjonalne ujęte są w nawiasy kwadratowe. Funkcje będące dokładnymi odpowiednikami funkcji PHP oznaczono symbolem  .

Niektóre funkcje, szczególnie statystyczne, wymagają operowania na konkretnych adresach e-mail będących w bazie. Z tego powodu nie będą działały poprawnie w niektórych miejscach systemu, gdzie nie można ustalić adresu lub nie znajduje się on w bazie. Przy opisie takiej funkcji znajdzie się uwaga „wymaga adresu".

mixed **array_search**(mixed *$igla*, array *$tablica*)

Funkcja zwraca indeks elementu, jeżeli wartość *$igla* znajduje się w tablicy *$tablica*, lub *false*, gdy jej nie znajdzie. Sprawdzanie jest ścisłe.

Przykład:

```
$tablica = array(2, 4, 6);
$x = 2;
$indeks = array_search($x, $tablica);
if ($indeks === false) {
      print("Liczba ".$x." nie znajduje się w zbiorze");
} else {
      print("Liczba ".$x." jest elementem nr ".($indeks+1)." zbioru");
}
```

Wynikiem będzie:

```
Liczba 2 jest elementem nr 1 zbioru
```

bool **barcode_ean13**(string *$kod*)

Funkcja służy do generowania kodów kreskowych w standardzie EAN13. Argument przekazywany funkcji musi zawierać 13 cyfr. Wywołana funkcja w miejscu swego wywołania wstawia znacznik HTML `<img src="odnośnik_do_grafiki" />` który spowoduje wyświetlenie w wiadomości e-mail grafiki przedstawiającej kod kreskowy o rozmiarach 109x70 pikseli. Jeżeli kod kreskowy był błędny, funkcja zwróci *false*, w innym przypadku *true*.

Przykład:

```
barcode_ean13("1234567890123");
```

string **bigregistry_get**(string **$rejestr**)

Odpowiednik funkcji registry_get() ale dla rejestrów typu BIG.

bool **bigregistry_set**(string *$rejestr,* string *$wartosc*)

Odpowiednik funkcji registry_set() ale dla rejestrów typu BIG.

bool **bigregistry_unset**(string *$rejestr*)

Odpowiednik funkcji registry_unset() ale dla rejestrów typu BIG.

number **campaign_clicked**(number *$nr_kampanii*)

Funkcja zwraca liczbę kliknięty ch wiadomości (nie liczbę kliknięty ch linków), jaką odnotowano dla bieżącego adresu e-mail w trakcie kampanii *$nr_kampanii*.

UWAGA: wymaga adresu.

Przykład:

```
if (campagain_clicked(7) > 0) {
      ok();
} else {
      system_skipmessage();
}
```

W tym przykładzie, w trakcie wysyłki zostaną pominięte adresy osób, które nie kliknęły żadnego linka podczas kampanii 7.

number **campaign_ctr**(number $nr_kampanii)

Funkcja zwraca stosunek kliknięty ch wiadomości do liczby wysłanych wiadomości na bieżący adres e-mail w trakcie kampanii $nr_kampanii. Wynik jest liczbą dziesiętną z zakresu od 0 do 1.

UWAGA: wymaga adresu.

Przykład:

```
if (campagain_ctr(7) >= 0.2) {
      ok();
} else {
      system_skipmessage();
}
```

W tym przykładzie, w trakcie wysyłki zostaną pominięte adresy osób, których stosunek kliknięty ch wiadomości do liczby wysłanych na bieżący adres podczas kampanii 7 był niższy niż 20%.

number **campaign_opened**(number *$nr_kampanii*)

Funkcja zwraca liczbę otwarcie, jaką zanotowano dla bieżącego adresu e-mail w trakcie kampanii *$nr_kampanii*.

UWAGA: wymaga adresu.

Przykład:

```
if (campagain_opened(7) > 0) {
      ok();
} else {
      system_skipmessage();
}
```

W tym przykładzie, w trakcie wysyłki zostaną pominięte adresy osób, które nie otwarły wiadomości e-mail podczas kampanii 7.

number **campaign_openrate**(number *$nr_kampanii*)

Funkcja zwraca stosunek otwartych wiadomości do liczby wysłanych wiadomości na bieżący adres e-mail w trakcie kampanii *$nr_kampanii*. Wynik jest liczbą dziesiętną z zakresu od 0 do 1.

UWAGA: wymaga adresu.

Przykład:

```
if (campagain_ctr(7) >= 0.33) {
      ok();
} else {
      system_skipmessage();
}
```

W tym przykładzie, w trakcie wysyłki zostaną pominięte adresy osób, których stosunek otwartych wiadomości do liczby wysłanych na bieżący adres podczas kampanii 7 był niższy niż 33%.

number **campaign_sent**(number *$nr_kampanii*)

Funkcja zwraca liczbę maili, jaką wysłano na bieżący adres e-mail w trakcie kampanii *$nr_kampanii*.

UWAGA: wymaga adresu.

Przykład:

```
if (mailing_campagain_sent(7) == 0) {
```

```
	ok();
} else {
	system_skipmessage();
}
```

W tym przykładzie, w trakcie wysyłki zostaną pominięte adresy osób, do których wysłano e-mail podczas kampanii 7.

number **campaign_total_clicked**(number *$nr_kampanii*)

Funkcja zwraca łączną liczbę kliknięty wiadomości (nie liczbę kliknięty linków), jaką odnotowano w trakcie kampanii *$nr_kampanii*.

number **campaign_total_ctr**(number *$nr_kampanii*)

Funkcja zwraca stosunek kliknięty wiadomości do liczby wysłanych wiadomości w trakcie kampanii *$nr_kampanii*. Wynik jest liczbą dziesiętną z zakresu od 0 do 1.

Przykład:

```
if (campagain_ctr(7) >= campagain_total_ctr(7)) {
	ok();
} else {
	system_skipmessage();
}
```

W tym przykładzie, w trakcie wysyłki zostaną pominięte adresy osób, których stosunek kliknięty wiadomości do liczby wysłanych na bieżący adres podczas kampanii 7 był niższy od średniej kampanii.

number **campaign_total_opened**(number *$nr_kampanii*)

Funkcja zwraca łączną liczbę otwarć, jaką zanotowano w trakcie kampanii *$nr_kampanii*.

number **campaign_total_openrate**(number *$nr_kampanii*)

Funkcja zwraca stosunek otwartych wiadomości do liczby wysłanych wiadomości w trakcie kampanii *$nr_kampanii*. Wynik jest liczbą dziesiętną z zakresu od 0 do 1.

number **campaign_total_sent**(number *$nr_kampanii*)

Funkcja zwraca łączną liczbę maili, jaką wysłano w trakcie kampanii *$nr_kampanii*.

number **count(**array *$tablica* [, number *$tryb*]**)**

Funkcja zlicza liczbę elementów w tablicy *$tablica*. Jeżeli jako opcjonalny argument *$tryb* podane zostanie 1, funkcja zliczać będzie elementy rekurencyjnie liczbę elementów (w przypadku tablic zagnieżdżonych).

Przykład:

```
$tablica = array(0, 3, 6, 9, 'a', '');
print(count($tablica));
```

Wynikiem będzie: `6`

string **date(**string *$format* [, number *$znacznik_czasu*]**)**

Funkcja *date()* zwraca czas sformatowany zgodnie z zawartością zmiennej *$format*. Funkcja jest identyczna z funkcją *date()* języka PHP. Jeżeli nie podamy opcjonalnego argumentu *$znacznik_czasu*, funkcja przyjmie aktualny uniksowy znacznik czasu (taki, jaki zostałby zwrócony przez funkcję *time()*). Jeżeli podany *$znacznik_czasu* nie będzie liczbą, funkcja zwraca *false*.

Aby otrzymać odpowiednio sformatowaną datę, należy użyć jednego lub więcej znaków z poniższej tabelki jako parametru *$format*.

Oryginalna tabela pochodzi ze strony http://www.php.net/manual/pl/function.date.php

Poniższych znaków używa się jako tekstu w parametrze `$format`

Zawartość parametru `$format`	Opis	Przykład zwróconej wartości
Dzień	---	---
d	Dzień miesiąca, 2 cyfry z wiodącymi zerami	*01* do *31*
D	Tekstowy opis angielskiej nazwy dnia, trzy litery	*Mon* kończąc na *Sun*
j	Dzień miesiąca bez zer wiodących	*1* do *31*
l (mała litera 'L')	Pełen angielski opis dnia tygodnia	*Sunday* aż do *Saturday*
N	Liczbowa forma dnia tygodnia, zgodna z normą ISO-8601	*1* (dla poniedziałku) aż do *7* (dla niedzieli)
S	Angielski przyrostek porządkowy dla dnia miesiąca, 2 litery	*st*, *nd*, *rd* lub *th*. Dobrze wygląda w połączeniu z *j*

Zawartość parametru `$format`	Opis	Przykład zwróconej wartości
w	Liczbowa forma dnia tygodnia	*0* (dla niedzieli) aż do *6* (dla soboty)
z	Dzień roku (zaczynając od 0)	*0* aż do *365*
Tydzień	---	---
W	Numer tygodnia w roku, zgodny z normą ISO-8601, Tygodnie rozpoczynają poniedziałki	Przykład: *42* (42. tydzień roku)
Miesiąc	---	---
F	Pełen angielski opis miesiąca, taki jak January czy March	*January* aż do *December*
m	Liczbowa forma miesiąca, z zerami wiodącymi	*01* aż do *12*
M	Krótki, angielski opis miesiąca, trzy litery	*Jan* a do *Dec*
n	Liczbowa forma miesiąca, bez zer wiodących	*1* aż do *12*
t	Liczba dni w danym miesiącu	*28* do *31*
Rok	---	---
L	Informacja o tym, czy rok jest przestępny	*1* jeśli rok jest przestępny, *0* w przeciwnym wypadku
o	Numer roku, zgodny z normą ISO-8601. Zwraca to taką samą wartość jak *Y*, z takim wyjątkiem, że numer tygodnia ISO (*W*) należy do poprzedniego lub następnego roku, niż rok użyty w tym miejscu.	Przykłady: *1999* lub *2003*
Y	Pełna liczbowa forma roku, 4 cyfry	Przykłady: *1999* lub *2003*
y	Dwie cyfry reprezentujące rok	Przykłady: *99* lub *03*
Czas	---	---

Zawartość parametru `$format`	Opis	Przykład zwróconej wartości
a	Pora dnia – dwie małe litery (przed/po południu) (ang. Ante/Post meridiem)	*am* lub *pm*
A	Pora dnia – dwie duże litery (przed/po południu) (ang. Ante/Post meridiem)	*AM* lub *PM*
B	Swatch Internet Time	*000* aż do *999*
g	Godzina, w formacie 12-godzinnym, bez zer wiodących	*1* aż do *12*
G	Godzina, w formacie 24-godzinnym, bez zer wiodących	*0* aż do *23*
h	Godzina, w formacie 12-godzinnym, z zerami wiodącymi	*01* aż do *12*
H	Godzina, w formacie 24-godzinnym, z zerami wiodącymi	*00* aż do *23*
i	Minuty z zerami wiodącymi	*00* do *59*
s	Sekundy z zerami wiodącymi	*00* aż do *59*
u	Mikrosekundy	Przykład: *54321*
Strefa czasowa	---	---
e	Identyfikator strefy czasowej	Przykłady: *UTC*, *GMT*, *Europe/Zagreb*
I (duże i)	Informacja o tym, czy czas jest letni	*1* jeśli czas jest letni, *0* w przeciwnym razie
O	Różnica z czasem Greenwich (GMT) w godzinach	Przykład: *+0200*
P	Różnica z czasem Greenwich (GMT) z dwukropkiem pomiędzy godzinami i minutami	Przykład: *+02:00*
T	Skrót dla strefy czasowej	Przykłady: *EST*, *MDT* ...
Z	Różnica dla strefy czasowej w sekundach. Wyrównanie to jest zawsze ujemne dla stref położonych na zachód od południka 0 oraz dodatnie dla tych leżących na wschód od niego.	*-43200* aż do *50400*

Zawartość parametru *$format*	Opis	Przykład zwróconej wartości
Pełna Data/Czas	---	---
c	Data w standardzie ISO 8601	2004-02-12T15:19:21+00:00
r	Data sformatowana zgodnie z RFC 2822	Przykład: *Thu, 21 Dec 2000 16:01:07 +0200*
U	Sekundy liczone od ery UNIX-a (1 stycznia 1970 00:00:00 czasu Greenwich - GMT)	Zobacz także funkcję time()

Inne znaki umieszczone w łańcuchu formatującym zostaną przez parser przepisane.

Przykłady:

Przykład	***Wynik***
date("Y-m-d H:i:s")	2010-02-23 13:12:34
date("l d F Y")	Tuesday 23 February 2010
date("g:i a")	1:12 pm
date("Y-m-d H:i:s", time()+3600)	2010-02-23 14:12:34

bool **empty**(mixed *$zmienna*)

Funkcja sprawdza, czy przekazany parametr jest „pusty”. Funkcja zwróci *true,* jeżeli *$zmienna* nie została wcześniej zdefiniowana lub posiada jedną z wartości: "" (pusty łańcuch); 0 (zero – jako liczba bądź łańcuch "0"); *null*; *false*; array() (pusta tablica).

Przykład:

```
$imie = get_val("prop1");
if (empty($imie)) {
      print("Drogi nieznajomy");
} else {
      $imie_wolacz = pl_imiewolacz($imie);
      print("Drogi ".$imie_wolacz);
}
```

array **explode**(string *$rozdzielenie,* string *$lancuch* [, number *$limit*])

Funkcja zwraca tablicę, której elementami są fragmenty podzielonego łańcucha $lancuch. Podział następuje w miejscach, gdzie w *$lancuch* zostanie znaleziony *$rozdzielenie*. Jeżeli *$rozdzielenie* nie zostanie znaleziony w *$lancuch*, tablica będzie zawierała tylko jeden element – cały łańcuch *$lancuch.*
Jeżeli zostanie użyty trzeci parametr *$limit* i będzie większy od zera, *$lancuch* zostanie podzielony na maksymalnie *$limit* elementów (ostatni element zwracanej tablicy będzie zawierał resztę łańcucha). Jeżeli $limit będzie mniejszy od zera, funkcja zwróci tablicę bez -*$limit* ostatnich elementów (w szczególności może zwrócić pustą tablicę). *$limit* równe 0 jest traktowane jakby było 1.

Przykład:

```
$lancuch = "To jest test";
$tablica = explode(" ",$lancuch);
print_r($tablica);
```

Wynikiem będzie:

```
Array
(
      [0] => To
      [1] => jest
      [2] => test
)
```

string **get_url**(string *$url* [, number *$extra* [, string *$encoding*]])

Funkcja pobiera treść (np. stronę www) znajdujący się pod podanym jako pierwszy parametr funkcji adresem URL. Drugi (opcjonalny) parametr funkcji pozwala na ustawienie dodatkowych operacji na pobieranej treści. *$extra* może zawierać wartości (traktowane bitowo):

- 1 – przekodowanie treści na prawidłowe wewnętrzne kodowanie systemu (UTF-8). Włączenie tej opcji powoduje, że treść zostanie przekodowana „w locie”. Jeżeli pobierana jest strona www, funkcja spróbuje odczytać kodowanie dokumentu na podstawie znacznika <meta> strony. Jeżeli nie będzie to możliwe, należy podać kodowanie dokumentu jako trzeci parametr funkcji *$encoding*.
  Opcja powinna być zawsze używana!
- 2 – napraw HTML. Użycie tej opcji spowoduje, że system będzie usiłował „naprawić” pobierany dokument HTML, tak aby spełniał wymagania systemu SARE dla newsletterów. Opcja powinna być zawsze używana w przypadku pobierania dokumentów HTML (stron www)! W przypadku innych dokumentów może być wyłączona, ze względu na to, że modyfikuje treść.
- 4 – próbuj zamienić relatywne odnośniki na bezwzględne. W przypadku pobierania strony www z

adresu URL, funkcja będzie próbowała zamienić względne odnośniki do grafik czy innych stron www na bezwzględne, tak aby wysłana później strona jako newsletter poprawnie wyświetlała się.

Domyślnie, jeżeli nie zostanie podana wartość drugiego parametru, funkcja przyjmie domyślną wartość 7 (1 + 2 + 3), czyli włączone są wszystkie trzy opcje.

Funkcja zwraca pobraną treść lub *false* w przypadku niepowodzenia.

UWAGA: Funkcja nie działa w kontekście adresu e-mail (w miejscach, gdzie jest wymagany adres e-mail – patrz dodatek B). Zwraca wtedy *false*.

Przykład:

```
$txt = 'Kliknij aby zobaczyć newsletter %www_version%';
$html = get_url('http://www.sare.pl/');
if ($html !== false) {
      newsletter_create($txt, $html, 'Newsletter '.date("Y-m-d"), 'Pobrano ze strony
www.sare.pl');
}
```

string **get_val**(string *$cecha*)

Funkcja pobiera z bazy cechę skojarzoną z bieżącym adresem e-mail. Argumentem dla get_val() jest nazwa cechy adresata. Dostępne są:

- *email* – bieżący adres e-mail,
- *gsm* – przypisany do adresu numer GSM,
- *status* – status adresu w systemie:

status	opis
2	Wypisany ręcznie przez operatora systemu SARE
3	Status nadawany przez operatora po wyrażeniu chęci wypisania się przez odbiorcę
4	Odbiorca zgłasza chęć wypisania, np. przez kliknięcie w link wypisujący
5	Adres nie został w żaden sposób potwierdzony, nie wysłano maila administracyjnego
6	Adres zapisany w bazie i oczekujący na potwierdzenie, wysłano mail administracyjny
7	Adres zablokowany przez operatora
8	Adres zapisany i potwierdzony

- *name* – pole Nazwa,
- *comment* – komentarz,
- *entry* – data wpisania adresu do bazy (w formacie Y-m-d H:i:s – patrz opis funkcji *date()*),
- *optedout* – ostatnia data zgłoszenia chęci wypisania się z bazy (w formacie Y-m-d H:i:s); jeżeli nie było

takiego zdarzenia, to pole zawiera 0000-00-00 00:00:00,

- *source* – pochodzenie adresu; 1 oznacza wpisanie ręczne; wartości od 10 do 99 oznaczają dodanie przez interfejs; wartości >100 oznaczają kolejne importy,
- *mailtype* – wybrany przez adresata rodzaj typ maila: "txt" lub "txt,html",
- *sent* – liczba wysłanych na ten adres maili (w ramach wysyłek mailingów),
- *smssent* – liczba wysłanych SMS-ów,
- *warns* – liczba ostrzeżeń przy dostarczaniu newslettera,
- *addresserrors* – liczba błędów związanych z adresem przy dostarczaniu newsletterów,
- *servererrors* – liczba błędów związanych z serwerem przy dostarczaniu newsletterów,
- *prop1* do *prop9* – cechy podstawowe,
- inne cechy zdefiniowane w bazie samodzielnie przez użytkownika.

Jeżeli cecha samodzielnie zdefiniowana przez użytkownika ma taką samą nazwę jak cechy systemowe, to zwrócona zostanie cecha zdefiniowana przez użytkownika.

UWAGA: wymaga adresu.

Przykład:

```
$imie = get_val("prop1");
$wpis = get_val("entry");
$miasto = get_val("miasto");
print("Witaj ".$imie."! Twoje miasto to ".$miasto." i jesteś z nami od ".$wpis);
```

Wynikiem będzie podobny do:

```
Witaj Adam! Twoje miasto to Warszawa i jesteś z nami od 2010-02-11 13:45:03
```

bool **in_arra**y(mixed *$igla*, array *$tablica*)

Funkcja zwraca *true,* jeżeli wartość *$igla* znajduje się w tablicy *$tablica,* lub *false,* gdy jej nie znajdzie. Sprawdzanie jest ścisłe.

Przykład:

```
$tablica = array(2, 4, 6);
$x = 2;
if (in_array($x, $tablica)) {
      print("Liczba ".$x." znajduje się w zbiorze");
}
```

Wynikiem będzie:

```
Liczba 2 znajduje się w zbiorze
```

number **intval**(mixed *$liczba [,* number *$baza]*)

Funkcja zwraca część całkowitą przekazanej liczby *$liczba*.

Opcjonalny parametr *$baza* służy jako baza dla konwersji liczb (tylko kiedy pierwszy argument jest typu string). Domyślnie jest to 10.

Przykład:

```
$a = 3;
$b = 2;
$c = intval(($a+$b)/2);
print("Część całkowita średniej liczb ".$a." i ".$b." wynosi ".$c);
```

Wynikiem będzie:

```
Część całkowita średniej liczb 3 i 2 wynosi 2
```

string **lastchar**(string *$lancuch*)

Funkcja zwraca ostatni znak przekazanego parametru.

Przykład:

```
$imie = "Barbaro";
$ostatnialitera = lastchar($imie);
print($ostatnialitera);
```

Wynikiem będzie: `0`

bool **mailing_bounced**(number *$nr_wysylki*)

Funkcja służy do sprawdzenia, czy bieżący adres ma odnotowany zwrot w wysyłce *$nr_wysyłki*. Jeżeli odnotowano zwrot, funkcja zwróci *true*, w innym przypadku *false* (również jeżeli nie wysłano do tej osoby e-maila).

UWAGA: wymaga adresu.

Przykład:

```
if (mailing_bounced(14)) {
      ok();
} else {
      system_skipmessage();
}
```

W tym przykładzie, w trakcie wysyłki zostaną pominięte adresy osób, do których wysłano e-mail podczas wysyłki 14 i zanotowano dla nich zwrot.

bool **mailing_clicked**(number *$nr_wysylki*)

Funkcja służy do sprawdzenia, czy bieżący adres ma odnotowane kliknięcie w wysyłce *$nr_wysyłki*. Jeżeli odnotowano kliknięcie, funkcja zwróci *true*, w innym przypadku *false*.

UWAGA: wymaga adresu.

Przykład:

```
if (mailing_clicked(14)) {
      ok();
} else {
      system_skipmessage();
}
```

W tym przykładzie, w trakcie wysyłki zostaną pominięte adresy osób, które nie kliknęły wiadomości z wysyłki 14.

bool **mailing_clicked_link**(number *$nr_wysylki,* number *$nr_linku*)

Funkcja służy do sprawdzenia, czy bieżący adres ma odnotowane kliknięcie w wysyłce *$nr_wysyłki* w link o identyfikatorze *$nr_linku*. Jeżeli odnotowano kliknięcie, funkcja zwróci *true*, w innym przypadku *false*.

UWAGA: wymaga adresu.

Przykład:

```
if (mailing_clicked_link(14, 133)) {
      ok();
} else {
      system_skipmessage();
}
```

W tym przykładzie, w trakcie wysyłki zostaną pominięte adresy osób, które nie kliknęły w link o identyfikatorze 133 w wiadomości z wysyłki 14.

array **mailing_info_get**(number *$nr_wysylki*)

Funkcja zwraca tablicę zawierająca ogólne informacje o przeprowadzonej (lub zaplanowanej) wysyłce *$nr_wysylki*. Jeżeli nie zostaną znalezione informacje o wysyłce, funkcja zwróci *false*. Zwracana tablica jest indeksowana z wykorzystaniem poniższych kluczy. Pola zawierają odpowiednio:

- *newsletter* – numer kreacji newslettera w systemie;
- *campaign* – numer kampanii, do której przypisano wysyłkę (0 – jeżeli do żadnej);
- *groups* – tablica zawierająca numery grup, do których została skierowana wysyłka. Numery odpowiadają numerom grup w momencie wysyłki i nie muszą odpowiadać obecnym (np. kiedy

została zmieniona ich nazwa);

- *date* – data, czas rozpoczęcia wysyłki w formacie *timestamp*;
- *encoding* – kodowanie wysłanego newslettera;
- *from* – pole From (nadawca) wysyłki;
- *replyto* – pole Reply-To wysyłki;
- *subject* – pole Subject (tytuł) wysyłki;
- *format* – format, w jakim została wysłana wiadomość: *txt* lub *txt,html*;
- *format_txt* – format, w jakim została wysłana wiadomość do osób, które zostały oznaczone jako odbiorcy chcący otrzymywać tylko wiadomości tekstowe: *txt* lub *txt,html*;
- *use_name* – zawiera 1, jeżeli wysyłkę przeprowadzono z zaznaczoną opcją „Używaj pola 'Nazwa' jako adresata", w innym przypadku 0;
- *trace_openings* – 1, jeżeli włączone było śledzenie otwarć, bądź 0, jeżeli wyłączone;
- *trace_clicks* – 1, jeżeli włączone było śledzenie kliknięć, bądź 0, jeżeli wyłączone;
- *attached_images* – 1, jeżeli mailing był wysyłany z dołączonymi grafikami, bądź 0, jeżeli z grafikami na serwerze;
- *filterSQL* – numer przypisanego filtra SQL;
- *filterSS* – numer przypisanego filtra SAREscript;
- *GA_string* – łańcuch doklejany do śledzonych linków (np. Google Analitics);

Przykład:

```
$mailing = mailing_info_get(123);
print_r($mailing);
```

Przykładowy wynik może być podobny do poniższego:

```
Array
(
    [newsletter] => 123
    [nr_kampanii] => 2
    [groups] => Array
        (
            [0] => 8
            [1] => 9
        )
    [encoding] => ISO-8859-2
    [format_txt] => txt
    [format] => txt,html
    [from] => "SARE - Newsletter" <info@sare.pl>
    [replyto] => mailing@sare.pl
    [subject] => Newsletter informacyjny
    [use_name] => 1
    [GA_string] =>
    [filterSQL] =>
    [filterSS] =>
    [date] => 1265776218
```

```
    [campaign] => 0
    [trace_openings] => 1
    [trace_clicks] => 1
    [attached_images] => 0
)
```

bool **mailing_opened**(number *$nr_wysylki*)

Funkcja służy do sprawdzenia, czy bieżący adres ma odnotowane otwarcie w wysyłce *$nr_wysyłki*. Jeżeli odnotowano otwarcie, funkcja zwróci *true*, w innym przypadku *false*.

UWAGA: wymaga adresu.

Przykład:

```
if (mailing_opened(14)) {
      ok();
} else {
      system_skipmessage();
}
```

W tym przykładzie, w trakcie wysyłki zostaną pominięte adresy osób, które nie otwarły wiadomości z wysyłki 14.

number **mailing_planned_count**([number *$ile_godzin*])

Funkcja zwraca liczbę wszystkich maili zaplanowanych do wysyłki na bieżący adres e-mail. Jeżeli został podany opcjonalny parametr *$ile_godzin*, zwrócona zostanie liczba maili, których wysyłkę planuje się w ciągu najbliższych *$ile_godzin* godzin. Do zaplanowanych wysyłek wliczane są wyłącznie wysyłki zwykłe czasowe.

UWAGA: wymaga adresu.

Przykład:

```
if ((mailing_sent_count(7*24)+mailing_planned_count() <=5) {
      ok();
} else {
      system_skipmessage();
}
```

W tym przykładzie, w trakcie wysyłki mail zostanie wysłany do osób, które w ciągu ostatnich siedmiu dni, łącznie z wysyłkami dopiero zaplanowanymi, nie otrzymały więcej niż 5 maili.

string **mailing_send**(array *$params* [, number *$kiedy*])

Funkcja przeprowadza wysyłkę mailigu. Wszystkie niezbędne parametry wysyłki należy przekazać w tablicy będącej pierwszym parametrem funkcji.

Opcjonalny parametr *$kiedy* pozwala na wysyłkę z opóźnieniem. Czas przekazuje się w postaci *timestampu* (zwracanego np. przez funkcję *time()*). Przykładowo, aby przeprowadzić wysyłkę za godzinę, można do wartości zwracanej przez time() dodać 3600 sekund (patrz przykład).

W przypadku niepowodzenia zwracana jest wartość *false*. Przekazywana tablica wykorzystuje poniższe klucze. Pola muszą zawierać odpowiednio:

- *newsletter* – numer istniejącej kreacji newslettera w systemie;

Opcjonalnie mogą zostać użyte pola opisane przy funkcji *mailing_send_test(),*oraz dodatkowo:

- *campaign* – nazwa lub numer kampanii, do której przypisać wysyłkę;
- *filterSQL* – numer filtra SQL;
- *filterSS* – numer filtra SAREscript;
- *ga_utm_source* – parametr *utm_source* dla Google Analitics;
- *ga_utm_medium* – parametr *utm_medium* dla Google Analitics;
- *ga_utm_campaign* – parametr *utm_campaign* dla Google Analitics;

UWAGA: Funkcja nie działa w kontekście adresu e-mail (w miejscach, gdzie jest wymagany adres e-mail – patrz dodatek B). Zwraca wtedy *false*.

Przykład:

```
$parametry["newsletter"] = 6;
$parametry["subject"] = "Tytuł";
$parametry["from"] = '"Redakcja newslettera" <newsletter@sare.pl>';
mailing_send_test("test@sare.pl", $parametry);
mailing_send_test($parametry, time()+3600);
```

mixed **mailing_send_test**(string *$email_testowy*, array *$params*)

Funkcja przeprowadza wysyłkę testową na adres *$email_testowy.* Wszystkie niezbędne parametry wysyłki należy przekazać w tablicy będącej drugim parametrem funkcji.

W przypadku niepowodzenia zwracana jest wartość false.

Przekazywana tablica wykorzystuje poniższe klucze. Pola muszą zawierać odpowiednio:

- *newsletter* – numer istniejącej kreacji newslettera w systemie;

Dodatkowo mogą zostać użyte poniższe pola:

- *subject* – pole Subject (tytuł) wysyłki. Domyślnie puste;
- *encoding* – kodowanie wysłanego newslettera. Kodowanie musi być zgodne z dostępnymi w systemie SARE (np. UTF-8, ISO-8859-2). Domyślnie UTF-8;
- *attached_images* – wysyłka będzie „z dołączonymi grafikami” zamiast „z grafikami na serwerze”,

jeżeli pole będzie miało wartość *true* (lub wartość może zostać zrzutowana na *bool true,* np. *1*). Domyślnie *false*;

- *from* – pole From (nadawca) wysyłki. Ple musi być zgodne z formatem:

  `"Nazwa nadawcy" <prawidlowy@email.nadawcy>`

  Nazwa nadawcy musi znajdować się w cudzysłowach (nie apostrofach!) i być oddzielona od właściwego adresu białym znakiem (np. spacją). Jeżeli nazwa nadawcy ma być pominięta, należy pominąć również cudzysłowy. Adres e-mail musi być umieszczony pomiędzy znakami < oraz >. Jeżeli w tablicy parametrów nie będzie przekazanej wartości *from* bądź będzie niepoprawna składniowo, pole From wysyłki przyjmie wartość domyślną, ustawioną w preferencjach systemu SARE.
- *replyto* – pole `Reply-To` wysyłki. Wartością powinien być poprawny składniowo adres e-mail. Domyślnie pole `Reply-To` wysyłki nie jest ustawiane;
- *use_name* – jeżeli wysyłka ma zostać przeprowadzona z wykorzystaniem pola Nazwa, jako adresata wiadomości (dodatkowo obok adresu w polu To wysyłki), pole powinno mieć wartość *true* (lub wartość może zostać zrzutowana na *bool true,* np. *1*). Jeżeli pole będzie miało wartość *false* (lub wartość może zostać zrzutowana na *bool false,* np. *0*), pole Nazwa nie zostanie użyte w opisany wcześniej sposób. Domyślnie, jeżeli w tablicy parametrów nie znajdzie się pole *use_name,* użyte zostanie ustawienie opcji „Używaj pola 'Nazwa' jako adresata” w preferencjach systemu SARE.
- *format* – format, w jakim ma zostać wysłana wiadomość. Domyślnie jest to *txt,html*. Aby wysłać tylko część tekstową, pole musi posiadać wartość *txt*;
- *format_txt* – format, w jakim ma zostać wysłana wiadomość do osób, które zostały oznaczone jako odbiorcy chcący otrzymywać tylko wiadomości tekstowe. Domyślnie jest to *txt*. Aby wysłać do nich również część HTML i tekstową, pole musi posiadać wartość *html,txt* ;
- *trace_openings* – aby zablokować śledzenie otwarć newsletterów, pole powinno mieć wartość *0* (*number*) lub *false*. Każda inna wartość (lub brak tego pola) spowoduje domyślne zachowanie, czyli dołączenie kodów śledzących do newslettera;
- *trace_clicks* – aby zablokować śledzenie kliknięć w linki (nie przerabiać oryginalnych linków), pole powinno mieć wartość *0* (*number*) lub *false*. Każda inna wartość (lub tego pola) spowoduje domyślne zachowanie, czyli śledzenie kliknięć;

UWAGA: Funkcja nie działa w kontekście adresu e-mail (w miejscach, gdzie jest wymagany adres e-mail – patrz dodatek B). Zwraca wtedy *false*.

Przykład:

```
$parametry["newsletter"] = 6;
$parametry["subject"] = "Tytuł";
$parametry["from"] = '"Redakcja newslettera" <newsletter@sare.pl>';
$mailing = mailing_send_test("test@sare.pl", $parametry);
```

bool **mailing_sent**(number *$nr_wysylki*)

Funkcja służy do sprawdzenia, czy na bieżący adres został wysłany mail podczas wysyłki *$nr_wysyłki*. Jeżeli został wysłany, funkcja zwróci *true*, w innym przypadku *false*. Sprawdzenie nie wystarcza do przyjęcia, że mail dotarł do odbiorcy.

UWAGA: wymaga adresu.

Przykład:

```
if (mailing_sent(14)) {
      ok();
} else {
      system_skipmessage();
}
```

W tym przykładzie, w trakcie wysyłki zostaną pominięte adresy osób, do których wysłano e-mail podczas wysyłki 14.

number **mailing_sent_count**([number *$ile_godzin*])

Funkcja zwraca liczbę wszystkich maili wysłanych na bieżący adres e-mail. Jeżeli został podany opcjonalny parametr *$ile_godzin*, zwrócona zostanie liczba maili wysłanych w trakcie ostatnich *$ile_godzin* godzin. Do wysłanych wliczane są wyłącznie wysyłki zwykłe oraz prośby o potwierdzenie rejestracji. Domyślnie system SARE przechowuje informacje o 200 ostatnich zrealizowanych wysyłkach.

UWAGA: wymaga adresu.

Przykład:

```
if (mailing_sent_count(7*24) <=5) {
      ok();
} else {
      system_skipmessage();
}
```

W tym przykładzie, w trakcie wysyłki zostaną pominięte adresy osób, do których wysłano w ciągu ostatnich siedmiu dni (7 dni razy 24 godziny) więcej niż 5 maili.

number **mailing_total_clicked**(number *$nr_wysylki*)

Funkcja zwraca liczbę wszystkich klikniętych maili podczas wysyłki *$nr_wysyłki*.

number **mailing_total_ctr**(number *$nr_wysyłki*)

Funkcja zwraca stosunek klikniętych wiadomości do liczby wysłanych wiadomości w trakcie wysyłki *$nr_wysylki*. Wynik jest liczbą dziesiętną z zakresu od 0 do 1.

number **mailing_total_opened**(number *$nr_wysylki*)

Funkcja zwraca liczbę wszystkich otwartych maili podczas wysyłki *$nr_wysyłki*.

number **mailing_total_openrate(**number *$nr_wysyłki***)**

Funkcja zwraca stosunek otwartych wiadomości do liczby wysłanych wiadomości w trakcie wysyłki *$nr_wysylki*. Wynik jest liczbą dziesiętną z zakresu od 0 do 1.

number **mailing_total_sent**(number *$nr_wysylki*)

Funkcja zwraca liczbę wszystkich wysłanych maili podczas wysyłki *$nr_wysyłki*.

number **newsletter_create** (string *$txt,* [string *$html,* [string *$nazwa*, [string *$opis*]]])

Funkcja służy do stworzenia kreacji newslettera. Pierwszym parametrem jest treść części tekstowej *$txt*. Parametr jest wymagany, jednak treść może być pusta. Jeżeli kreacja ma zawierać część html, przekazujemy ją jako drugi parametr *$html*. Kolejny parametr *$nazwa* to nazwa kreacji (newslettera), pod jaką będzie widoczny w systemie. Jeżeli nie zostanie podany (lub będzie pusty), newsletter zostanie zapisany pod nazwą „SAREscript *data*", gdzie *data* to czas jego stworzenia. Ostatni parametr *$opis* to opis kreacji, który będzie widoczny w systemie.

Funkcja zwraca numer, pod jakim został zapisany newsletter, lub *false* w przypadku niepowodzenia.

Przykład:

```
$txt = 'Kliknij aby zobaczyć newsletter %www_version%';
$html = get_url('http://www.sare.pl/');
if ($html !== false) {
	$parametry["newsletter"] = newsletter_create($txt, $html);
	if ($parametry["newsletter"] != false) {
		$parametry["subject"] = "Tytuł";
		$parametry["from"] = '"Redakcja newslettera" <newsletter@sare.pl>';
		mailing_send_test("test@sare.pl", $parametry);
		mailing_send($parametry, time()+3600);
	}
}
```

bool **not**(bool *$param*)

Funkcja zwraca zanegowaną wartość parametru $param. Jest odpowiednikiem operatora **!** (negacja).

Przykład:

```
if (not(false)) {
      print("Warunek zawsze prawdziwy");
} else {
      print("Warunek zawsze fałszywy");
}
```

Wynikiem będzie:

```
Warunek zawsze prawdziwy
```

bool **ok**()

Funkcja nie przyjmuje żadnych argumentów i zawsze zwraca *true.* Praktycznym wykorzystaniem może być poprawienie czytelności kodu.

Przykład:

```
if (sare_in_group(1)) {
      ok();
} else {
      system_skipmessage();
}
```

W tym przykładzie, w trakcie wysyłki zostaną pominięte adresy nie należące do grupy 1.

string **pl_imiewolacz**(string *$imie*)

Funkcja wyszukuje w bazie polskich imienia *$imie* przekazanego jako parametr funkcji i jeżeli odnajdzie imię, to zwraca formę wołaczową tego imienia. Jeżeli nie znajdzie, zwraca przekazany parametr.

Przykład:

```
$imie = get_val("prop1");
$imie_wolacz = pl_imiewolacz($imie);
print("Drogi ".$imie_wolacz);
```

mixed **pl_kody**(string *$kod* [, string *$kod2* [, number *$odleglosc*]])

Funkcja pozwala na manipulację polskimi kodami pocztowymi. Może ona posiadać od jednego do

trzech parametrów.

Jeżeli podamy tylko jeden parametr *$kod*, funkcja zwróci tablicę zawierającą następujące pola:

- kod – kod pocztowy (taki sam jak *$kod*);
- miasto – miejscowość, w której znajduje się urząd pocztowy o kodzie *$kod*;
- województwo – województwo, w którym znajduje urząd pocztowy posiadający kod *$kod*;

Jeżeli podamy również drugi parametr *$kod2*, funkcja zwróci przybliżoną odległość wyrażoną w kilometrach, pomiędzy środkami miejscowości będących siedzibami głównych urzędów pocztowych o podanych kodach.

Możliwe jest również podanie trzeciego parametru *$odleglosc.* Funkcja zwróci *true,* jeżeli odległość pomiędzy środkami miejscowości będących siedzibami głównych urzędów pocztowych o kodach *$kod* i *$kod2* jest mniejsza lub równa *$odleglosc*, oraz *false* w przeciwnym wypadku.

Funkcja zawsze zwróci *null*, jeżeli któryś z kodów nie występuje w bazie.

Przykład:

```
$kod = get_val('kod_pocztowy');
if (pl_kody($kod, '44-200', 100)) {
      ok();
} else {
      system_skipmessage();
}
```

W podanym przykładzie wysyłka zostanie przeprowadzona do osób, które zamieszkują nie dalej niż 100 km od miejscowości o kodzie 44-200.

string **pl_slownie**(number *$liczba*)

Funkcja zamienia liczbę (całkowitą) na odpowiednik słowny.

Przykład:

```
$kwota = 2534;
$slownie = pl_slownie($kwota);
print("Słownie: ".$slownie);
```

Wynikiem będzie:

```
Słownie: dwa tysiące pięćset trzydzieści cztery
```

number **print**(mixed *$arg*)

Funkcja print() jest podstawową funkcją umożliwiającą zwrócenie danych ze scriptspotu. Jej

argumentem jest łańcuch znaków (bądź liczba), który ma zostać wyświetlony. Funkcja zawsze zwraca liczbę 1.

Przykład:

```
print("Dzień dobry");
```

Wynikiem będzie:

```
Dzień dobry
```

mixed **print_r**(mixed *$arg* [, bool *$return*])

Funkcja print_r() jest odpowiednikiem funkcji PHP o takiej samej nazwie. Zwraca w czytelnej postaci wartość argumentu (np. zawartość całej tablicy, która w przypadku użycia funkcji print() wymagałaby wskazania konkretnego indeksu). Praktyczne zastosowanie jest ograniczone do debugowania skryptów.

Jeżeli zostanie podany drugi argument i będzie miał wartość true, funkcja zamiast wyświetlać wynik, zwróci go. W innym przypadku funkcja zawsze zwróci *true*.

Przykład:

```
$array = array(1,2,3);
print_r($array);
```

Wynikiem będzie:

```
Array
(
      [0] => 1
      [1] => 2
      [2] => 3
)
```

number **rand**(number *$min*, number *$max*)

Funkcja zwracająca losową liczbę całkowitą z przedziału określonego przez *$min* i *$max*. Największe dopuszczalne *$max* to 2147483647.

Przykład:

```
$rzut_kostka = rand(1,6);
print("Wylosowano: ".$rzut_kostka);
```

Wynikiem będzie liczba losowa od 1 do 6.

string **registry_get**(string *$rejestr*)

Funkcja odczytuje wartość rejestru o nazwie *$rejestr* i zwraca jego wartość. Jeżeli rejestr o takiej nazwie nie istnieje, zwracany jest pusty łańcuch.

Przykład:

```
$rejestr = registry_get("mój rejestr");
```

bool **registry_set**(string *$rejestr,* string *$wartosc*)

Funkcja zapisuje *$wartosc* do rejestru o nazwie *$rejestr* i zwraca jego wartość. Zwraca *true,* jeżeli aktualizacja rejestru przebiegnie poprawnie, lub *false,* gdy wystąpi problem.

Przykład:

```
registry_set("mój rejestr", "To jest zawartość zapisywana do rejestru");
```

bool **registry_unset**(string *$rejestr*)

Funkcja usuwa rejestr o nazwie *$rejestr.* Zwraca *true,* jeżeli był taki rejestr i jego usunięcie przebiegło poprawnie, lub *false,* gdy nie było takiego rejestru bądź wystąpił problem z jego usunięciem.

Przykład:

```
registry_unset("mój rejestr");
```

array **sare_groups**()

Funkcja zwraca tablicę zawierającą grupy, do których należy bieżący adres. Nie są przekazywane do niej żadne parametry.

UWAGA: wymaga adresu.

Przykład:

```
print_r(sare_groups());
```

Wynik może być podobny do:

```
Array
(
[0] => 2
[1] => 9
[2] => 14
[3] => 16
)
```

bool **sare_in_group**(number *$grupa*)

Funkcja zwraca *true,* jeżeli bieżący adres należy do grup $grupa, lub *false* w innym przypadku.

Przykład:

```
if (sare_in_group(1) and sare_in_group(4)) {
      print("Adres należy do grupy 1 i 4");
}
```

Wynikiem będzie:

```
Adres należy do grupy 1 i 4
```

bool **set_val**(string *$cecha*, string *$wartosc*)

Funkcja ustawia w bazie cechę skojarzoną z bieżącym adresem e-mail. Argumentem dla set_val() jest nazwa cechy adresata oraz wartość, jaka ma zostać zapisana.

UWAGA: wymaga adresu.

UWAGA! Funkcja pozwala operować tylko na cechach samodzielnie stworzonych (rozszerzona struktura danych). Cechy 1-9 są chronione.

Zwraca *true,* jeżeli aktualizacja cechy przebiegnie poprawnie, lub *false,* gdy wskazana cecha nie jest zdefiniowana.

Przykład:

```
$imie = "Kasia";
set_val("zwrot", "Pani ".$imie);
```

Wynikiem działania będzie zapisanie w cesze „zwrot” zdania: `Pani Kasia`

bool **sms_received**(number *$nr_wysylki_sms*)

Funkcja służy do sprawdzenia, czy dla bieżącego adresu (numeru SMS) zarejestrowano odebranie wiadomości SMS pochodzącej z wysyłki SMS *$nr_wysyłki_sms*. Jeżeli został wysłany, funkcja zwróci *true*, w innym przypadku *false*.

UWAGA: wymaga adresu.

Przykład:

```
if (sms_received(6)) {
      ok();
} else {
      system_skipmessage();
}
```

W tym przykładzie, w trakcie wysyłki zostaną pominięte adresy osób, dla których odnotowano odebranie

SMS z wysyłki nr 6.

bool **sms_sent**(number *$nr_wysylki_sms*)

Funkcja służy do sprawdzenia, czy na bieżący adres (numer SMS) został wysłany SMS podczas wysyłki SMS *$nr_wysyłki_sms*. Jeżeli został wysłany, funkcja zwróci *true*, w innym przypadku *false*.

UWAGA: wymaga adresu.

Przykład:

```
if (sms_sent(6)) {
      ok();
} else {
      system_skipmessage();
}
```

W tym przykładzie, w trakcie wysyłki zostaną pominięte adresy osób, do których wysłano SMS podczas wysyłki SMS nr 6.

array **str_split**(string *$lancuch* [, number *$dlugosc*])

Funkcja konwertuje łańcuch *$lancuch* na tablicę. Każdy znak będzie zamieniony na element tablicy wynikowej.

Jeżeli zostanie podany drugi opcjonalny parametr *$dlugosc*, elementami tablicy zostaną fragmenty łańcucha *$lancuch* o maksymalnej długości *$dlugosc* znaków. Jeżeli argument $dlugosc będzie mniejszy od 1, funkcja zwróci *false*.

Przykład:

```
$lancuch = "test";
$tablica = str_split($lancuch);
print_r($tablica );
```

Wynikiem będzie:

```
Array
(
      [0] => t
      [1] => e
      [2] => s
      [3] => t
)
```

array **str_repeat**(string *$lancuch*, number *$ile*)

Funkcja zwraca *$lancuch* powtórzony *$ile* razy.

Przykład:

```
$lancuch = "ox";
print(str_repeat($lancuch, 5));
```

Wynikiem będzie:

```
oxoxoxoxox
```

string **str_pad**(string *$lancuch*, number *$dlugosc* [, string *$wypelnienie* [, number *$typ*]])

Funkcja zwraca *$lancuch* uzupełniony do długości *$dlugość* znaków, za pomocą *$wypelnienie*. Jeżeli *$dlugosc* będzie mniejsza lub równa liczbie znaków łańcucha *$lancuch*, nie zostanie on zmieniony. Jeżeli argument $wypelnienie nie zostanie podany, domyślnie będzie to spacja.

Czwartym argumentem funkcji jest sposób dopełniania. W zależności od jego wartości *$lancuch* będzie dopełniany:

0 – doklejaj z lewej strony;

1 – doklejaj z prawej strony;

2 – doklejaj z obu stron.

Jeżeli nie zostanie podany argument *$typ*, domyślnie *$lancuch* dopełniany będzie z prawej strony.

Przykład:

```
$lancuch = "abc";
print(str_pad($lancuch, 5, "-"));
```

Wynikiem będzie:

```
abc--
```

string **strtolower**(string *$lancuch*)

Funkcja zwraca *$lancuch* ze wszystkimi wielkimi literami zamienionymi na małe.

Przykład:

```
$hobby = "FILATELISTYKA 2000";
$hobby = strtolower($hobby);
print($hobby);
```

Wynikiem będzie:

```
filatelistyka 2000
```

string **substr**(string *$lancuch*, number *$start* [, number *$length*])

Funkcja zwraca ciąg znaków o długości *$length* łańcucha *$lancuch*, począwszy od znaku *$start.* Pozycja pierwszego znaku liczona jest od zera 0 (nie od jedynki). Jeżeli parametr *$length* został pominięty,

funkcja zwróci fragment łańcucha od *$start* do końca.

Przykład:

```
$zdanie = "Ala ma kota";
$kto = substr($zdanie, 0, 3);
print("Kto: ".$kto);
$co = substr($zdanie, 7);
print("Co: ".$co);
```

Wynikiem będzie:

```
Kto: Ala
Co: kota
```

string **substr_count**(string *$igla*, string *$lancuch* [, number *$start* [, number *$length*]])

Funkcja zwraca liczbę wystąpień *$igla* w *$lancuch*. Opcjonalnie można podać trzeci parametr *$start*, wskazujący, od którego znaku ma rozpocząć się przeszukiwanie.

Wyszukiwanie jest *case sensitive* (wielkość liter jest rozróżniana).

Przykład:

```
$zdanie = "Ala ma kota";
$ileA = substr_count("A",$zdanie);
print("W zdaniu liter A jest: ".$ileA);
$ilea = substr_count("a",$zdanie);
print("W zdaniu liter a jest: ".$ilea);
```

Wynikiem będzie:

```
W zdaniu liter A jest: 1
W zdaniu liter a jest: 3
```

void **system_setfrom**(string *$adres_email* [, string *$nadawca*])

Funkcja ustawia pole From w wysyłanej wiadomości. Jeżeli zawiera tylko jeden parametr *$adres_email,* to ustawiany jest tylko adres e-mail. Drugi parametr *$nadawca* to nazwa nadawcy.

UWAGA: Działa tylko podczas wysyłki mailingu.

Przykład:

```
system_setfrom('sarescript@sare.pl', 'SAREscript');
```

void **system_setsubject**(string *$subject*)

Funkcja ustawia pole Subject (Temat) w wysyłanej wiadomości.

UWAGA: Działa tylko podczas wysyłki mailingu.

Przykład:

```
system_setsubject('Tytuł maila');
```

void **system_skipmessage**()

Jest to funkcja specjalna kontrolująca wykonywanie SAREscriptu. Funkcja zawsze powoduje przerwanie dalszego przetwarzania SAREscriptu w wiadomości. Jeżeli funkcja zostanie użyta w wysyłce właściwej, jej wywołanie spowoduje pominięcie bieżącego odbiorcy.

Przykład:

```
if ($wiek<18) {
      system_skipmessage();
}
```

number **time**()

Funkcja zwraca bieżący uniksowy znacznik czasu. Przydatny w połączeniu z funkcją *date()*.

Przykład:

```
print("Teraz jest ".date("H:i", time()).", a za godzinę będzie ".date("H:i",
time()+3600));
```

Wynik może być podobny do:

```
Teraz jest 11:27, a za godzinę będzie 12:27
```

string **trim**(string *$lancuch* [, string *$znaki*])

Funkcja usuwa „białe znaki” z początku i końca łańcuch *$lancuch*.

Dodatek B: Szczegóły wykonywania scriptspotów

SAREscript może zostać wykonany w kilku miejscach w systemie. Różnią się one obsługą niektórych funkcjonalności języka.

	Do działania wymagany adres e-mail	Modyfikacja danych w bazie	Osadzenie grafiki	system_ setfrom()	system_ setsubject()
Mailing > edytor – podgląd	●	-	●	-	-
Mailing > edytor – podgląd na email	●	-	●	-	-
Mailing > wysyłanie – mail testowy	●	-	○[1)]	●	●
Mailing > wysyłanie – testowanie filtrów	-	-	-	-	-
Mailing > wysyłanie – wysyłka właściwa					
- filtr zaawansowany	●	●	-	-	-
- w treści TXT	●	●	●	●	●
- w treści HTML	●	●	●	●	●
SAREscript > uruchom – wykonanie skryptów, w zależności od wybranego źródła danych:					
- brak	-	-	-	-	-
- wybrane dane	-	-	-	-	-
- wybrane adresy	●	●	-	-	-
- cała baza	●	●	-	-	-
SAREscript > uruchom – debugowanie skryptów, w zależności od wybranego źródła danych:					
- brak	-	-	-	-	-
- wybrane dane	-	-	-	-	-
- wybrane adresy	●	-	-	-	-
- cała baza	●	-	-	-	-
Raporty > mapa kliknięć	●	-	●	-	-

Podgląd WWW wiadomości	●		●	-	-
Scenariusze subskrypcji – wyświetlanie i wysyłanie obiektów	●	●	●	●	●
Interfejs SOAP – wywołanie metody Execute	-	●	-	-	-

● działa; - nieobsługiwane; ○[1)] działa tylko przy wysyłce „z dołączonymi grafikami"

W niektórych miejscach systemu (np. przy wysyłce wiadomości) istotna jest kolejność, w jakiej są wykonywane scriptspoty.

1. Mailing > wysyłanie-mail testowy

1. Wyszukanie linków w treści maila
2. Wykonanie SAREscript w treści TXT (patrz tabela powyżej)
3. Osadzenie wyniku działania scriptspotów TXT w mailu
4. Wyszukanie i podmiana znaczników personalizacyjnych (*%znacznik%*) w treści TXT
5. Wykonanie SAREscript w treści HTML (patrz tabela powyżej)
6. Osadzenie wyniku działania scriptspotów HTML w mailu
7. Wyszukanie i podmiana znaczników personalizacyjnych (*%znacznik%*) w treści HTML
8. Wysłanie maila

2. Mailing > wysyłanie – wysyłka właściwa

1. Wybranie adresów do wysyłki z zastosowaniem filtra prostego
2. Wykonanie dla każdego wybranego adresu e-mail filtra zaawansowanego (patrz tabela powyżej)
3. Wyszukanie linków w treści maila
4. Wyszukanie w treści maila i użycie znaczników <sare_subject>, <sare_from_f>, <sare_from_m>, <sare attachment>
5. Wykonanie SAREscript w treści TXT (patrz tabela powyżej)
6. Przekodowanie wyniku działania scriptspotów TXT na kodowanie docelowe
7. Osadzenie wyniku działania scriptspotów TXT w mailu
8. Jeżeli odbiorca chce odbierać HTML lub wymuszono wysyłkę HTML, wykonanie SAREscript w treści HTML (patrz tabela powyżej)
9. Przekodowanie wyniku działania scriptspotów HTML na kodowanie docelowe
10. Osadzenie wyniku działania scriptspotów HTML w mailu

11. Wyszukanie i podmiana znaczników personalizacyjnych (*%znacznik%*) w treści TXT
12. Wyszukanie i podmiana znaczników personalizacyjnych (*%znacznik%*) w treści HTML
13. Wysłanie maila

3. SAREscript > wykonaj – wykonanie lub debugowanie na wybranych adresach

1. Wybranie adresów z zastosowaniem filtra prostego
2. Wykonanie dla każdego wybranego adresu e-mail filtra zaawansowanego (patrz tabela powyżej)
3. Wykonanie SAREscript dla każdego z wybranych adresów (patrz tabela powyżej)

V. Narzędzia dodatkowe

1. Debugger kodu

System SARE umożliwia przetworzenie skryptu SAREscript w sposób ułatwiający wychwycenie potencjalnych błędów. Służy temu narzędzie zwane debuggerem.

W celu rozpoczęcia sesji debugowania należy wybrać w menu SAREscript > uruchom. Podobnie jak przy wykonywaniu skryptów mamy możliwość wyboru źródła danych dla sesji debugowania.
Jako źródło danych możemy wybrać:

- brak – sesja debugowania nie będzie korzystać z danych pobranych z adresów e-mail,
- wybrane dane – sesja debugowania wykorzysta schemat danych z formularza (dane są jedynie pobierane z przykładowego adresu, sama sesja nie ma związku z adresem),
- wybrane adresy – sesja debugowania wykorzysta dane kolejnych adresów w wybranych grupach po zastosowaniu (opcjonalnie) filtrów prostych i filtra zaawansowanego,
- cała baza – sesja debugowania wykorzysta dane kolejnych adresów w bazie (adresy w porządku alfabetycznym).

Debugowanie jest mechanizmem bezpiecznym dla bazy adresowej. W żadnym z powyższych schematów sesja debugowania nie spowoduje modyfikacji danych adresów.

Aby debugować skrypt, należy go wpisać w edytorze lub wybrać jeden z szablonów bądź gotowych skryptów zapisanych w koncie. Po zmianie funkcji z *Wykonaj skrypt* na *Debuguj skrypt* nastąpi zmiana interfejsu edytora. System przetwarza kod skryptu na potrzeby debugowania, co może spowodować zmianę układu linii skryptu w porównaniu do poprzedniego widoku. Od tego momentu istnieje możliwość oznaczenia punktu wstrzymania sesji debugowania (*break point*) w lewej kolumnie za pomocą przełączników. Zaznaczając przełącznik, spowodujemy, że sesja debugowania będzie kontynuowana do linii bezpośrednio poprzedzającej linię z zaznaczonym przełącznikiem.

Sesję debugowania uruchamiamy przyciskiem *Start*. Po wykonaniu lub zatrzymaniu wykonania w punkcie wstrzymania aktualny rezultat jest widoczny w oknie *Wynik*. Jeżeli skrypt nie spowoduje wypisania czegokolwiek na ekran (np. przez użycie funkcji print()), rezultat wykonania skryptu może być pusty, pomimo że skrypt wykonał się prawidłowo.

Jeżeli sesja debugowania została wstrzymana przy pomocy punktu wstrzymania, mamy możliwość jej kontynuowania za pomocą przycisków *continue* i *step*. Przycisk *continue* służy do wznowienia sesji do kolejnego punktu wstrzymania, natomiast użycie przycisku *step* spowoduje wykonanie kolejnej linii skryptu.

W momencie wstrzymania sesji istnieje możliwość modyfikowania wartości zmiennych (typy

proste i tablicowe) oraz typu zmiennych (typy proste). Mechanizm ten pozwala na kontynuowanie sesji debugowania z nowymi wartościami i typami zmiennych. W celu zmiany wartości lub typu zmiennej należy kliknąć na wartości zmiennej (w momencie gdy kursor jest w trybie edycji). Zmianę typu lub wartości zatwierdzamy przyciskiem *OK*. Następnie należy użyć przycisku *continue* lub *step* w celu kontynuowania sesji z nowymi wartościami/typami cech.

W momencie ukończenia wykonywania skryptu dla jednego adresu system zapyta, czy wykonywać skrypt dla kolejnych adresów (dla opcji cała baza lub wybrane adresy). Stan sesji debugowania między adresami jest resetowany, tj. zmienne jednego adresu nie wpływają na zmienne kolejnych. Można również anulować przetwarzanie kolejnego adresu – przycisk *next* umożliwiający kontynuowanie przetwarzania adresów pojawi się na pasku narzędzi obok przycisku *step*.