Bazele limbajului Swift

Swift este un nou limbaj de programare pentru dezvoltarea aplicațiilor iOS, macOS, watchOS și tvOS. Cu toate acestea, multe părți ale Swift vor fi similare cu cele din C și Objective-C.

Swift oferă propriile versiuni ale tuturor tipurilor fundamentale C și Obiectiv-C, inclusiv Int pentru numere întregi, Double și Float pentru valori în virgulă mobilă, Bool pentru valori booleene și String pentru date textuale. Swift oferă, de asemenea, versiuni puternice ale celor trei tipuri de colecții primare, Array, Set și Dictionary.

La fel ca și C, Swift folosește variabilele pentru stocarea și referirea la valori printr-un nume de identificare. Swift utilizează, de asemenea, extensiv variabile ale căror valori nu pot fi modificate. Acestea sunt cunoscute sub numele de constante și sunt mult mai puternice decât constantele în C. Constantele sunt folosite pe tot parcursul scrieri codului Swift pentru a face codul mai sigur și mai clar cu privire la intenția codului atunci când lucrați cu valori care nu trebuie să se schimbe.

În plus față de tipurile familiare, Swift introduce tipuri avansate care nu sunt găsite în Objective-C, cum ar fi tuplul. Tuplul vă permite să creați și să pasați mai departe grupuri de valori. Puteți folosi un tuplu pentru a returna mai multe valori dintr-o funcție ca o singură valoare.

Swift introduce de asemenea tipuri opționale care se ocupă de absența unei valori. Opționalele spun fie "există o valoare și este egală cu x" sau "nu există o valoare". Utilizarea opționalelor este similară cu utilizarea valorii nil atribuita unui pointer în Objective-C, dar aceștia lucrează pentru orice tip, nu doar pentru clase. Nu numai că opționalele sunt mai sigure și mai expresive decât pointerii nuli în Objective-C, ele sunt în centrul multor caracteristici ale lui Swift.

Swift este un limbaj sigur, ceea ce înseamnă că limbajul vă ajută să vă lămuriți cu privire la tipurile de valori pe care le poate utiliza codul. Dacă o parte din codul dvs. necesită un String, tipul de siguranță vă împiedică să îl transmiteți din greșeală unui Int. De asemenea, tipul de siguranță vă împiedică să transmiteți accidental un String opțional unei bucăți de cod care necesită un String ne-opțional. Siguranța de tip vă ajută să prindeți și să remediați erorile cât mai curând posibil în procesul de dezvoltare.

Constante și variabile

Constantele și variabilele asociază un nume (cum ar fi maximumNumberOfLoginAttempts sau welcomeMessage) cu o valoare a unui anumit tip (cum ar fi numărul 10 sau șirul "Hello"). Valoarea unei constante nu poate fi schimbată odată setată, în timp ce o variabilă poate fi setată la o valoare diferită în viitor.

Constantele și variabilele trebuie declarate înainte de a fi utilizate. Declarați constantele folosind cuvântul cheie *let* și variabilele folosind cuvântul cheie *var*. Iată un exemplu despre modul în care constantele și variabilele pot fi utilizate pentru a urmări numărul de încercări de conectare pe care le-a făcut un utilizator:

  
Figura 1: Definirea unei constante si a unei variabile

Acest cod poate fi citit ca:

"Declarați o nouă constantă numită maximumNumberOfLoginAttempts și dați-i o valoare de 10. Apoi, declarați o nouă variabilă numită currentLoginAttempt și dați-i o valoare inițială de 0."

În acest exemplu, numărul maxim de încercări de conectare permise este declarat ca o constantă, deoarece valoarea maximă nu se modifică niciodată. Contorul actual de încercare de autentificare este declarat ca o variabilă, deoarece această valoare trebuie incrementată după fiecare încercare de conectare nereușită.

Adnotări de tip

Puteți oferi o adnotare de tip atunci când declarați o constantă sau o variabilă, pentru a fi mai clar cu privire la tipul de valori pe care constanta sau variabila o poate stoca. Scrieți o adnotare de tip plasând două puncte după numele constantei sau variabilei, urmat de un spațiu, urmat de numele tipului utilizat.

Acest exemplu oferă o adnotare de tip pentru o variabilă numită welcomeMessage, pentru a indica faptul că variabila poate stoca valorile String:

  
Figura 2: Declararea unei variabile specificând tipul

Cele două puncte din declarație înseamnă "... de tip ...", astfel încât codul de mai sus poate fi citit ca: "Declarați o variabilă numită welcomeMessage, care este de tip String." Variabila welcomeMessage poate fi acum setată la orice valoare de șir fără eroare.

Rareori va fi nevoie să scrieți adnotări de tip în practică. Dacă furnizați o valoare inițială pentru o constantă sau o variabilă în punctul definit, Swift poate deduce aproape întotdeauna tipul care va fi utilizat pentru acea constantă sau variabilă. În exemplul welcomeMessage de mai sus, nu este furnizată nici o valoare inițială, deci tipul variabilei welcomeMessage este specificat cu o adnotare de tip, mai degrabă decât dedus dintr-o valoare inițială.

Inferența tipului și siguranță de tip

Swift este un limbaj sigur pentru tip. Un limbaj sigur de tip vă încurajează să vă lămuriți cu privire la tipurile de valori cu care codul dvs. poate funcționa. Dacă o parte din codul dvs. necesită un String, nu îi puteți trece din greșeală un Int.

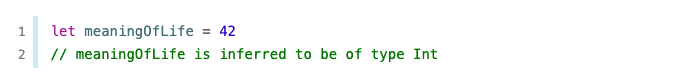
Deoarece Swift prezintă siguranța de tip, acesta efectuează verificări de tip atunci când compilează codul dvs. și scoate în evidență orice tip necompatibil ca și eroare. Acest lucru vă permite să prindeți și să remediați erorile cât mai curând posibil în procesul de dezvoltare.

Verificarea tipului vă ajută să evitați erorile atunci când lucrați cu diferite tipuri de valori. Cu toate acestea, acest lucru nu înseamnă că trebuie să specificați tipul fiecărei constante și variabile pe care le declarați. Dacă nu specificați tipul de valoare de care aveți nevoie, Swift utilizează inferența de tip pentru a elabora tipul corespunzător. Inferența de tip permite unui compilator să deducă automat tipul unei expresii în mod automat atunci când compilează codul dvs., pur și simplu examinând valorile pe care le furnizați.

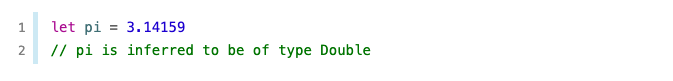
Din cauza inferenței de tip, Swift necesită mai puține declarații de tip decât limbajele precum C sau Objective-C. Constantele și variabilele sunt încă inscripționate în mod explicit, dar o mare parte din munca de specificare a tipului lor este făcută pentru dvs.

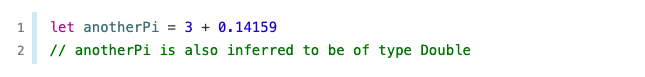
Inferența de tip este utilă în special atunci când declarați o constantă sau o variabilă cu o valoare inițială. Acest lucru se face deseori atribuindu-se o valoare literală constantă sau variabilă în punctul în care o declarați. (O valoare literală este o valoare care apare direct în codul dvs. sursă, cum ar fi 42 s 3.14159 în exemplele de mai jos.)

De exemplu, dacă atribuiți o valoare literală de 42 unei constante noi, fără a spune ce tip este, Swift deduce că doriți ca constantul să fie un Int, deoarece ați inițializat-o cu un număr care arată ca un întreg:

  
Figura 3: Inițializarea unei constante cu un literal

De asemenea, dacă nu specificați un tip pentru un literal în virgulă mobilă, Swift deduce că doriți să creați un Double:

  
Figura 4: Inițializarea unei constante cu un Double

  
Valoarea literală a lui 3 nu are nici un tip explicit în sine și astfel un tip de ieșire adecvat, și anume Double, este dedus din prezența unui literal în virgulă mobilă ca parte a adăugării.

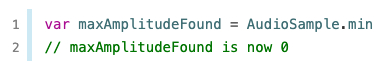
Aliasuri de tip

Aliasurile de tip definesc un nume alternativ pentru un tip existent. Definiți alias-uri de tip cu cuvântul cheie *typealias.*

Aliasurile de tip sunt utile atunci când doriți să vă referiți la un tip existent printr-un nume care este mai adecvat din punct de vedere contextual, cum ar fi atunci când lucrați cu date de o anumită dimensiune dintr-o sursă externă:



După ce definiți un alias de tip, puteți utiliza pseudonimul oriunde ați putea utiliza numele original:

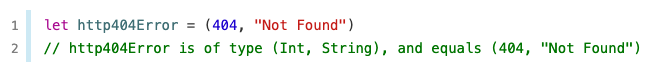


Aici, AudioSample este definit ca un alias pentru UInt16. Deoarece este un alias, apelul catre AudioSample.min este de fapt UInt16.min, care ofera o valoare initiala de 0 pentru variabila maxAmplitudeFound.

Tupluri

Tuplurile grupează valori multiple într-o singură valoare compusă. Valorile dintr-un tuplu pot fi de orice tip și nu trebuie să fie de același tip ca și celelalte.

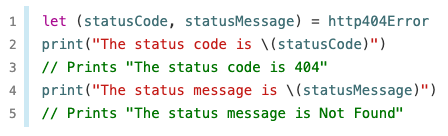
În acest exemplu, (404, "Not Found") este un tuplu care descrie un cod de stare HTTP. Un cod de stare HTTP este o valoare specială returnată de un server web ori de câte ori solicitați o pagină Web. Un cod de stare de 404 Nu a fost găsit este returnat dacă solicitați o pagină web care nu există.



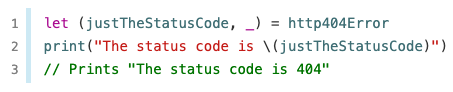
Tuplul (404, "Not Found") grupează un Int și un String pentru a da codului de stare HTTP două valori separate: un număr și o descriere care poate fi citită de om. Acesta poate fi descris ca "un tuplu de tip (Int, String)".

Puteți crea tupluri din orice permutare a tipurilor și pot conține cât mai multe tipuri diferite. Nu există nimic care să te oprească de a avea un tuplu de tip (Int, Int, Int) sau (String, Bool), sau chiar orice altă permutare pe care o doriți.

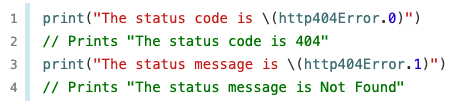
Puteți descompune conținutul unui tuplu în constante sau variabile separate, pe care le accesați ca de obicei:



Dacă aveți nevoie doar de unele dintre valorile tuplului, ignorați părți ale lui cu o subliniere (\_) atunci când descompuneți tuplul:



Alternativ, accesați valorile elementelor individuale dîntr-un tuplu folosind numerele index care încep de la zero:



Puteți numi elementele individuale într-un tuplu atunci când este il definiți:



Tuplurile sunt utile în special ca valori de întoarcere a funcțiilor. O funcție care încearcă să acceseze o pagină web ar putea să returneze tipul de tip (Int, String) pentru a descrie succesul sau eșecul accesării paginii. Prin returnarea unui tuplu cu două valori distincte, fiecare cu un tip diferit, funcția oferă informații mai utile despre rezultatul său decât dacă ar putea să returneze numai o singură valoare a unui singur tip.

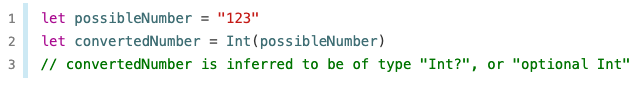
Tuplurile sunt utile pentru grupuri simple de valori asociate. Nu sunt potrivite pentru crearea unor structuri complexe de date. Dacă structura de date este probabil complexă, modelați-o mai degrabă ca o clasă sau o structură, decât ca un tuplu.

Opționale

Utilizați opționalele în situațiile în care o valoare poate fi absentă. Un opțional reprezintă două posibilități: Fie că există o valoare, și puteți să „despachetați” opționalul pentru a accesa acea valoare, fie nu există o valoare deloc.

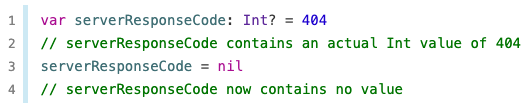
Conceptul de opțional nu există în C sau în Objective-C. Cel mai apropiat lucru în Objective-C este abilitatea de a întoarce o valuare nulă dintr-o metodă care altfel ar întoarce un obiect, prin nil semnificând "absența unui obiect valid". Totuși, aceasta funcționează numai pentru obiecte - nu funcționează pentru structuri, tipuri primitive din C sau valori ale unei enumerații. Pentru aceste tipuri, metodele Objective-C returnează de obicei o valoare specială (cum ar fi NSNotFound) pentru a indica absența unei valori. Această abordare presupune că apelantul metodei știe că există o valoare specială pentru cazurile în care nu se gasește o valoare și are obligația de a compara rezultatul metodei cu acea valoare specială. Opționalele Swift vă permit să indicați absența unei valori pentru orice tip, fără a fi nevoie de constante speciale.

Iată un exemplu despre modul în care opționalele pot fi utilizate pentru a arăta înspre absenței unei valori. Tipul Int din Swift are un constructor care încearcă să convertească o valoare String într-o valoare Int. Cu toate acestea, nu fiecare șir poate fi convertit într-un număr întreg. Șirul "123" poate fi convertit în valoarea numerică 123, dar șirul "hello, world" nu are o valoare numerică evidentă pentru a se converti la.

 Deoarece constructorul s-ar putea să eșueze, acesta returnează un Int opțional, mai degrabă decât un Int. Un opțional Int este scris ca Int?, nu Int. Semnul de întrebare indică faptul că valoarea pe care o conține este opțională, ceea ce înseamnă că ar putea conține o anumită valoare Int sau ar putea să nu conțină nicio valoare. (Nu poate conține nimic altceva, cum ar fi o valoare Bool sau o valoare String. Este fie un Int, fie nu este nimic.)

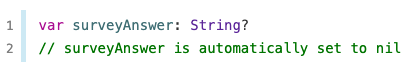
nil

Se poate seta o variabilă opțională la o stare fără valoare atribuindu-i valoarea specială nil:



Nu puteți folosi *nil* cu constante și variabile non-opționale. Dacă o constantă sau o variabilă din codul dvs. trebuie să funcționeze cu absența unei valori în anumite condiții, declarați-o întotdeauna ca o valoare opțională de tipul corespunzător.

Dacă definiți o variabilă opțională fără a furniza o valoare implicită, variabila este setată automat la nil pentru dvs.:

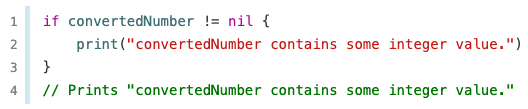


Nilul din Swift nu este același cu nilul în Objective-C. În Objective-C, nil este un pointer către un obiect inexistent. În Swift, nil nu este un pointer - este absența unei valori de un anumit tip. Opționalele de orice tip pot fi setate la nil, nu doar cele care sunt instanțe ale unor clase.

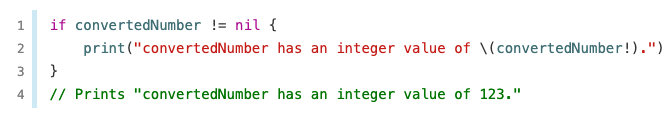
Instrucțiunea *if* și „despachetarea” forțată

Puteți utiliza o instrucțiune if pentru a afla dacă un opțional conține o valoare prin compararea opțiunii opționale cu nil. Efectuați această comparație cu operatorul "egal cu" (==) sau cu operatorul "diferit de" (!=).

Dacă un opțional are o valoare, el este considerat a fi "diferit de" nil:



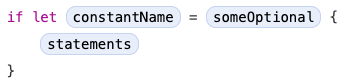
Odată ce sunteți sigur că opționalul conține o valoare, puteți accesa valoarea sa de bază adăugând un semn de exclamare (!) la sfârșitul numelui opționalului. Semnul de exclamare spune: "Știu că aceast opțional are cu siguranță o valoare; utilizați-o". Aceasta este cunoscută sub denumirea de despachetare forțată a valorii opționale:



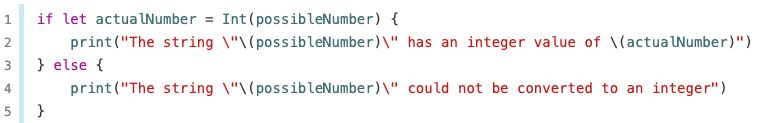
Încercarea de a folosi „!” pentru a accesa o valoare opțională inexistentă va face programul să încheie forțat. Asigurați-vă întotdeauna că opționalul conține o valoare non-nil înainte de a utiliza „!” pentru a-i despacheta valoarea.

Optional Binding

Utilizați Optional Binding pentru a afla dacă un opțional conține o valoare și, dacă da, pentru a face ca această valoare să fie disponibilă ca o constantă sau variabilă temporară. Optional binding-ul poate fi utilizat cu instrucțiunile *if* și *while* pentru a verifica valoarea din interiorul unui opțional și pentru a extrage acea valoare într-o constantă sau variabilă, ca parte a unei singure acțiuni. Optional binding-ul are următoarea structură:



Exemplul de mai sus poate fi rescris folosind optional binding în felul următor:



Acest cod poate fi citit ca: "Dacă Int-ul opțional returnat de Int(possibleNumber) conține o valoare, setați o nouă constantă numită actualNumber cu valoarea conținută de opțional."

Dacă conversia are succes, constanta actualNumber devine disponibilă pentru a fi utilizată în prima ramură a instrucțiunii if. Acesta a fost deja inițializată cu valoarea conținută în opțional, deci nu este nevoie să utilizați „!” la sfârșitul numelui constantei pentru a accesa valoarea sa. În acest exemplu, actualNumber este pur și simplu utilizat pentru a imprima rezultatul conversiei.

Puteți utiliza atât constante, cât și variabile cu optional binding. Dacă doriți să manipulați valoarea actualNumber în prima ramură a instrucțiunii if, ați putea scrie *if var actualNumber* și valoarea conținută în opțional ar fi disponibilă ca o variabilă în loc de constantă.

Opționale implicit „despachetate”

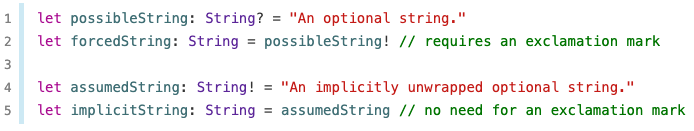
După cum este descris mai sus, opționalele arată că unei constante sau unei variabile îi este permis să aibă "nicio valoare". Opționalele pot fi verificate cu o instrucțiune if pentru a vedea dacă există o valoare și pot fi „despachetate” cu optional binding pentru a accesa valoarea opțională dacă aceasta există.

Uneori este clar din structura unui program că un opțional va avea întotdeauna o valoare, după ce valoarea este setată pentru prima dată. În aceste cazuri, este util să eliminați necesitatea de a verifica și de a „despacheta” valoarea opționalului de fiecare dată când este accesat, deoarece se poate presupune în siguranță că are o valoare tot timpul.

Aceste tipuri de opționale sunt definite ca fiind opțional implicit „despachetate”. Un opțional implicit „despachetat” se scrie prin plasarea unui semn de exclamare (String!) în locul unui semn de întrebare (String?) după tipul pe care doriți să-l faceți opțional.

Opționalele implicit „despachetate” sunt utile atunci când se confirmă existența unei valori imediat după ce opționalul este definit pentru prima dată și se poate presupune că există în orice moment după aceea.

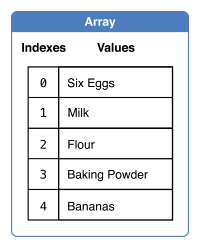
Un opțional implicit „despachetat” este un opțional normal, dar poate fi de asemenea folosit ca o valoare non-opțională, fără a fi nevoie să despachetați valoarea opțională de fiecare dată când este accesat. Următorul exemplu prezintă diferența de comportament dintre un șir opțional și un șir opțional implicit „despachetat” atunci când accesează valoarea din el ca un String explicit:

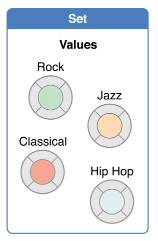
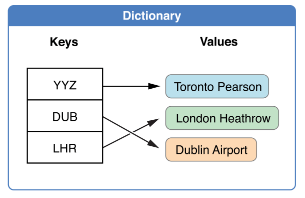


Vă puteți gândi la un opțional implicit „despachetat”, ca acordând permisiunea să fie „despachetat” automat ori de câte ori este folosit. În loc să introduceți un semn de exclamare după numele opționalului de fiecare dată când îl folosiți, plasați un semn de exclamare după tipul opțional când îl declarați.

Dacă un opțional implicit „despachetat” este nefolosit și încercați să accesați valoarea dinăuntrul lui, veți declanșa o eroare de runtime. Rezultatul este exact același ca și când ați plasa un semn de exclamare după un opțional obișnuit care nu conține o valoare.

Tipuri de colecții

 Swift oferă trei tipuri primare de colecții, cunoscute sub formă de Array, Set și Dictionary, pentru stocarea colecțiilor de valori. Array-urile sunt colecții ordonate de valori. Seturile sunt colecții neordonate de valori unice. Dicționarele sunt colecții neordonate de asocieri cheie-valoare.



Array-urile, Set-urile și Dicționarele în Swift sunt întotdeauna clare cu privire la tipurile de valori și chei pe care le pot stoca. Aceasta înseamnă că nu puteți introduce o valoare de tip greșit într-o colecție din greșeală. De asemenea, înseamnă că puteți avea încredere în tipul de valori pe care le veți obține dintr-o colecție. Ele sunt implementate precum colecții generice. Mai multe detalii vom vedea în capitolul de generice.

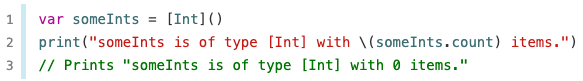
Dacă creați un Array, Set sau Dictionary și îl atribuiți unei variabile, colecția care este creată va fi mutabilă. Aceasta înseamnă că puteți schimba (sau muta) colecția după ce a fost creată prin adăugarea, eliminarea sau schimbarea elementelor din colecție. Dacă atribuiți Array, Set sau Dictionary la o constantă, acea colecție este imutabilă, iar dimensiunea și conținutul acesteia nu pot fi modificate ulterior.

Array (sau vector)

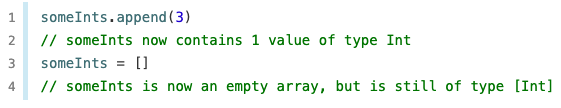
Un array stochează valori de același tip într-o listă ordonată. Aceeași valoare poate apărea într-un array de mai multe ori la poziții diferite.

Tipul unui array Swift este scris în întregime ca Array<Element>, unde Element este tipul de valori pe care array-ul îl stochează. De asemenea, puteți scrie tipul de array sub forma prescurtată [Element]. Deși cele două forme sunt identice din punct de vedere funcțional, forma preferată este cea prescurtată.

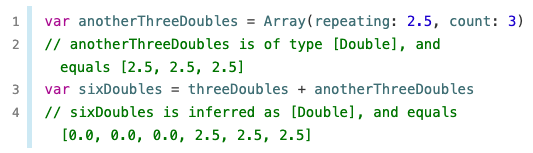
Se poate crea un array gol folosind următoarea sintaxă:



Alternativ, dacă contextul oferă deja informații de tip, cum ar fi un argument al funcției sau o variabilă sau o constantă deja declarată, puteți crea un array gol cu expresia literală de array gol, care este scrisă ca [] (o pereche goală de paranteze pătrate):



Puteți crea un array nou adăugând împreună două array-uri existente cu tipuri compatibile cu operatorul adițional (+). Tipul noului array este dedus din tipul celor două array-uri pe care le adăugați împreună:



De asemenea, puteți inițializa un array cu un array literal, care este o modalitate de prescurtată de a scrie una sau mai multe valori ca o colecție de tip array. Un array literal este scris ca o listă de valori, separate prin virgule, înconjurată de o pereche de paranteze pătrate:





Variabila shoppingList este declarată ca "un array de tip String", scris ca [String]. Deoarece aceast array are specificat tipul de valoare conținut, String, este permisă stocarea valorilor numai de tipul String. Aici, array-ul shoppingList este inițializat cu două valori String ("Eggs" și "Milk"), scrise într-un array literal.

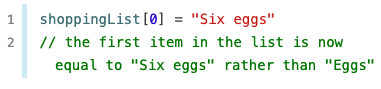
În acest caz, literalul array-ului conține două valori de tip String și nimic altceva. Acest lucru se potrivește cu tipul declarației variabilei shoppingList (un array care poate conține numai valori String), astfel încât alocarea literalului de array este permisă ca o modalitate de a inițializa shoppingList cu două elemente inițiale.

Accesul și modificarea unui array

Accesul și modificarea unui array se face pe bază de subscript. Accesul la o valoare din array utilizând subscriptul se face în felul următor, trecând index-ul valorii pe care doriți să o accesați în paranteze pătrate imediat după numele array-ului:

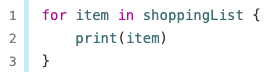
  
Primul element din matrice are un index-ul 0, nu 1. Array-urile în Swift sunt întotdeauna indexate incepând cu 0.

Subscriptul se poate folosi și pentru a modifica o valoare de la un anumit index:

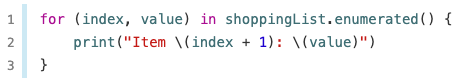


Interarea printr-un array

Se poate face printr-un for in:



Dacă aveți nevoie de indexul fiecărui element, precum și de valoarea acestuia, utilizați metoda enumerated() pentru a itera prin array. Pentru fiecare element din colectia, metoda enumerated() returnează un tuplu compus dintr-un întreg și elementul acelui index. Numerele întregi încep de la zero și sunt incrementate cu câte unul pentru fiecare element; dacă enumerați tot array-ul aceste numere întregi se potrivesc cu indicii elementelor. Puteți să descompuneți tuplul în constante sau variabile temporare ca parte a iterației:



Set

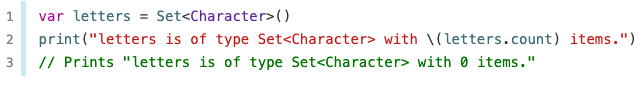
Un set stochează valori distincte de același tip într-o colecție fără o anume ordine. Puteți utiliza un set în locul unui array când ordinea articolelor nu este importantă sau când trebuie să vă asigurați că un element apare o singură dată.

Un tip trebuie să fie hashable pentru a fi stocat într-un set - adică tipul trebuie să ofere o modalitate de a calcula o valoare hash pentru el însăși. O valoare hash este o valoare int care este aceeași pentru toate obiectele care se compară egal, astfel încât dacă a == b, rezultă că a.hashValue == b.hashValue.

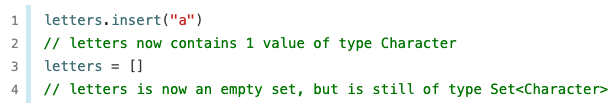
Toate tipurile de bază Swift (cum ar fi String, Int, Double și Bool) sunt hashable implicit și pot fi utilizate ca tipuri de valoare pentru un Set sau precum chei ale unui dicționar. Valorile cazurilor de enumerare fără valori asociate sunt, de asemenea, hashable implicit.

Puteți utiliza propriile tipuri ca tipuri de valoare pentru un Set sau precum chei ale unui dicționar, făcându-le să fie conforme cu protocolul Hashable din biblioteca standard Swift. Tipurile care sunt conforme cu protocolul Hashable trebuie să furnizeze o proprietate de tipul Int numită hashValue. Valoarea returnată de proprietatea hashValue a unui tip nu trebuie să fie aceeași pentru diferite execuții ale aceluiași program sau în diferite programe.

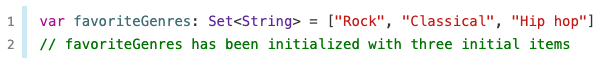
Puteți crea un set gol de anumit tip utilizând sintaxa constructorului:



În mod alternativ, dacă contextul oferă deja informații de tip, cum ar fi un argument de funcție sau o variabilă sau o constantă deja declarată, puteți crea un set gol, cu un array literal gol:



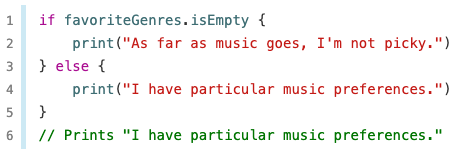
De asemenea, puteți inițializa un set cu un array literal, ca o modalitate scurtă de a scrie una sau mai multe valori ca o colecție de tip Set. Exemplul de mai jos creează un set numit favoriteGenres ce stochează valori de tip String:

 Variabila favoriteGenres este declarată ca "un set de valori String", scrisă ca Set<String>. Deoarece acest set are specificat ca tip de valoare String, este permisă numai stocarea valorilor String. Aici, setul favoriteGenres este inițializat cu trei valori String ("Rock", "Classical" și "Hip hop"), scrise într-un array literal.

Accesarea și modificarea unui Set

Accesul și modificarea unui set se face prin metodele și proprietățile sale.

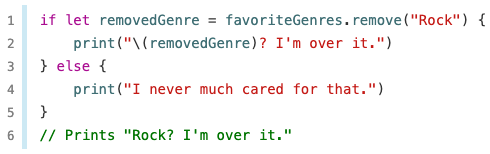
Utilizați proprietatea isEmpty ca o comandă rapidă pentru a verifica dacă proprietatea count este egală cu 0:



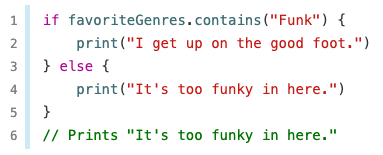
Puteți adăuga un element nou într-un set prin apelarea metodei insert(\_:) a setului:



Puteți elimina un element dintr-un set apelând metoda remove(\_:) a setului, care elimină elementul dacă este membru al setului și returnează valoarea eliminată sau nil dacă setul nu conține valoarea. În mod alternativ, toate elementele dintr-un set pot fi eliminate cu metoda removeAll().

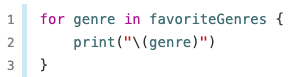


Pentru a verifica dacă un set conține un anumit element sau nu, se poate utiliza metoda contains(\_:).

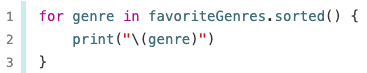


Iterarea printr-un Set

Puteți itera peste valorile dintr-un set cu o buclă for-in.



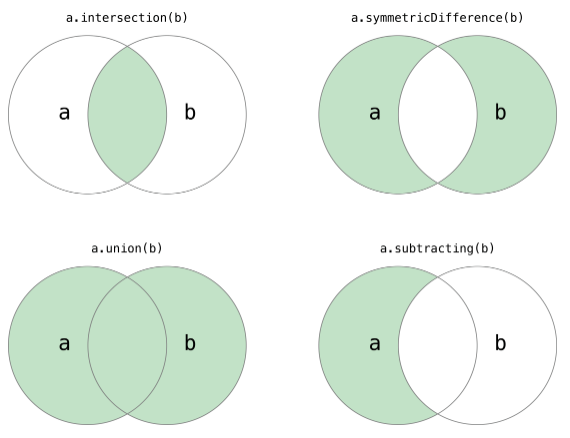
Tipul Set din Swift nu are o ordine definită. Pentru a itera peste valorile unui set într-o anumită ordine, utilizați metoda sorted(), care returnează elementele setului ca un array sortat folosind operatorul <.

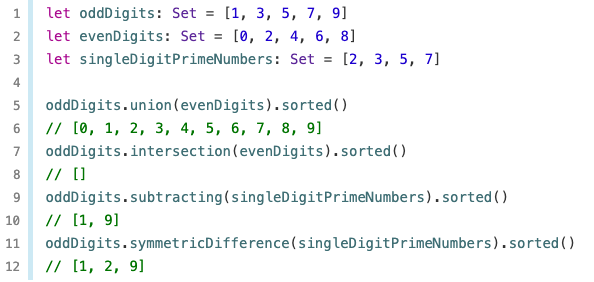


Puteți efectua eficient operațiile de Set fundamentale, cum ar fi combinarea a două seturi împreună, determinarea valorilor pe care două seturi le au în comun sau determinarea dacă două seturi conțin toate, unele sau nici una dintre aceleași valori. Set-ul din Swift este analogul mulțimii din matematică.

Ilustrația de mai jos descrie două seturi - a și b - cu rezultatele diferitelor operații de Set reprezentate de regiunile umbrite.

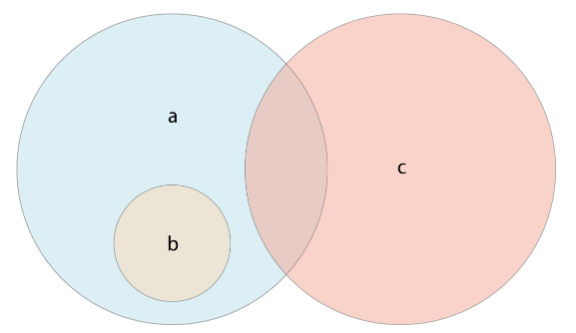
* Utilizați metoda intersection(\_:) pentru a crea un set nou, cu valori comune celor două seturi.
* Utilizați metoda symmetricDifference(\_:) pentru a crea un set nou cu valori în oricare dintre seturi, dar nu ambele.
* Utilizați metoda union(\_:) pentru a crea un set nou cu toate valorile din ambele seturi.
* Utilizați metoda de subtracting(\_:) pentru a crea un set nou cu valori care nu se află în setul specificat.





Calitatea de membru și egalitate

Ilustrația de mai jos descrie trei seturi - a, b și c - cu regiunile suprapuse reprezentând elemente partajate între seturi. Setul a este un superset al setului b, deoarece a conține toate elementele din b. În schimb, setul b este un subset al setului a, deoarece toate elementele din b sunt de asemenea conținute de a. Setul b și setul c sunt disjuncte unul cu celălalt, deoarece nu au elemente în comun.



* Utilizați operatorul "este egal" (==) pentru a determina dacă două seturi conțin aceleași valori.
* Utilizați metoda isSubset(of:) pentru a determina dacă toate valorile unui set sunt conținute în setul specificat.
* Utilizați metoda isSuperset(of:) pentru a determina dacă un set conține toate valorile dintr-un set specificat.
* Utilizați metodele isStrictSubset(of:) sau isStrictSuperset(of:) pentru a determina dacă un set este un subset sau superset, dar nu egal cu, un set specificat.
* Utilizați metoda isDisjoint(with:) pentru a determina dacă două seturi nu au valori comune.

