Fundamentals of Perl 6
From Zero to Scripting

By Jeffrey Goff, in conjunction with O'Reilly Media

What Is Perl 6?

- Perl's newest language family member
- Released Christmas 2015
- Dynamically typed
- Runs on multiple VMs

Where is it Used?

- System Administration
- Web Hosting
- Database Administration
- Gaming
- MIDI and Computer Music

Where Do I Find New Toys?

- http://modules.perl6.org
- https://github.com/tadzik/panda
- https://github.com/ugexe/zef

Where Do I Go For Help?

- http://www.perl6.org
- http://docs.perl6.org
- https://modules.perl6.org
- irc://irc.freenode.net #perl6

Perl 6 at the Command Line

```
$ perl6 -e'say “Hello, world!”;'
```

How To Express Yourself

```
$ perl6 -e'say “1 + 2 is “, 1 + 2;'
1 + 2 is 3
$ perl6 -e'say “1 + 2 is {1 + 2}”;'
1 + 2 is 3
```

How To Say Things

```
$ perl6 -e 'say “Hello, World!”;'
Hello, World!
$ perl6 -e 'say q{“Well...”, she said};'
“Well...”, she said
```

Perl 6 on the file system

```
#!/usr/bin/env perl6
say “Hello, world!”;
```

How to Store Things

- Singular variables
  - $thing – stores a single value such as 1 or “Hello”
  - &function – stores a function
- Plural variables
  - @cubicles – Stores a list of singular variables
  - %dictionary – Stores a hash table

Singular Variables

- Start with $ (or &, but that's for advanced users)
- Names start with Unicode alphabetic character
  - Yes, $Ω and $Ж are legal variable names
  - So are $double_07 and $MI-6
  - But not $1st or $1.81-Gigawatts
- Hold strings, numbers or objects
- Can have optional types

What Can You Store In Them?

- Numbers
  - Integers
  - Rationals
  - Complex
- Strings

  - Anything in Unicode like “∀a:a∉N”
- Objects
  - From classes or modules you create or load

How To Declare Them

```
#!/usr/bin/env perl6
my $year = 2017;
my Str $place = “Babylon 5”;
say “The year is $year”;
say “The place is $place”;
```

```
The year is 2017
The place is Babylon 5
```

Fundamental Arithmetic

- perl -E 'say 0.1 + 0.2 – 0.3'
  - +5.55111512312578e-17
- python -c 'print 0.1 + 0.2 – 0.3'
  - +5.55111512312578e-17
- ruby -e 'print 0.1 + 0.2 – 0.3'
  - +5.55111512312578e-17
- perl6 -e 'say 0.1 + 0.2 – 0.3'
  - 0

Optional Typing

$ perl6 -e'my $ship = “Red Dwarf”; say “on board $ship”;'

on board Red Dwarf

$ perl6 -e'my Str $ship = “Red Dwarf”; say “on board $ship”;'

on board Red Dwarf

Type Checking

```
$ perl6 -e'my $a = “Hello World”; say $a'
Hello World

$ perl6 -e'my Str $a = 123; say $a'
Type check failed in assignment to $a;
expected Str but got Int (123) in block <unit>
at -e line 1
```

String Interpolation

```
$ perl6 -e 'my Str $a = “World!”; say “Hello $a”;'
Hello World!

$ perl6 -e 'my Str $a = “World!”; say q{Hello $a};'
Hello $a
```

Comparisons

```
$ perl6 -e 'say 1 < 3;'
True
$ perl6 -e 'say 1 > 3;'
False
$ perl6 -e'say “Yes” if 1 < 3;'
Yes
```

Branching

```
my $answer = 17;
my $guess = 32;
if $answer == $guess {
    say “Got it right!”
} elsif $answer < $guess {
    say “Too low!”
}
```

Multi-Way Branching

```
if $answer < $guess {
    say “Guess is too high”
} elsif $answer > $guess {
    say “Guess is too low”
} else {
    say “Guess is just right!”
}
```

String comparison

```
my $answer = “animal”;
my $guess = “vegetable”;
if $answer eq $guess {
    say “Guessed $guess, got it right!”
} else {
    say “I guessed wrong”
}
```

Accepting User Input

```
my $name;
print “name> “;
$name = $*IN.get;
say “My name is $name”;


name> Larry
My name is Larry
```

How to Loop

```
my $done;
while not $done {
    my $guess;
    print “guess> “;
    $guess = $*IN.get;
    if $guess == 42 {
        $done = 1;
    }
}
```

Your First Exercise

Write a simple “Guess My Number” game in Perl 6.

You can create a “hidden” number with this statement:

```
my $hidden-number = 128.rand;
```

- Create a “hidden” number for the user to guess
- Loop until the user guesses correctly.
  - Let the user enter a guess (with my $guess = $*IN.get;)
  - If the guess is correct, print “You guessed it!” and exit.
  - If the guess is too low, print “Too low!” and give the user another chance
  - If the guess is too high, print “Too high!” and give the user another chance
  - Optional: If the user enters a negative value, print “Bye!” and exit.

One Way To Do It

```
my Int $target = 128.rand;
my $done;
while not $done {
    my $guess;
    print “your guess> “;
    $guess = $*IN.get;
    if $guess < 0 {
        say “bye!”;
        $done = 1;
    } elsif $guess < $target {
        say “Guess too low!”;
    } elsif $guess > $target {
        say “Guess too high!”
    } else {
        say “Just right!”;
        $done = 1;
   }
}
```

Advanced Text Comparison

- Partial matching – substrings
  - “double 007” ~~ / 7 /
- Anchored matching
  - “double 007” ~~ / ^ double /
- Generic matching
  - “double 007” ~~ / \w+ /

Plural Variables

- Start with @ or %
- @-variables hold 0+ singular values
  - my @x; @x[0] = 42;
- %-variables hold 0+ dictionary pairs
  - my %x; %x{'key'} = 'value';

A Little About Arrays

- Ordered data type
- Cells can contain almost anything
  - Including arrays, allowing recursive structures
  - Including hashes, allowing complex data types
- Dynamically allocated
- Built-in iterators
- Built-in list comprehension

Working with Lists

```
my @evens = 2, 4, 6;
say “The even numbers are”, @evens;
```

The even numbers are 2 4 6

```
say “The even numbers are really {@evens}”;
```

The even numbers are really 2 4 6

Slicing and Dicing

```
my @evens = 2, 4, 6;
say “The last even is @evens[2]”;
```

The last even is 6

```
say “The first two evens are @evens[0,1]”;
```

The first two evens are 2 4

Mixed Arrays

```
my @mixed = 1, 'apple', 2, 'banana';
say “Fruit salad is @mixed[1,3]”;
```

Fruit salad is apple banana

Operations On Lists

```
my @a = 1, 3, -5, 0, 2;
say “max: “, max @a;
say “min: “, min @a;
say “average: “, sum( @a ) / @a.elems;
```

```
max: 3
min: -5
average: 0.2
```

List Operations

```
my @a = 1, -3, 5, 6, 5;
say “sorted: “, sort @a;
say “backwards: “, reverse @a;
```

```
sorted: -3 1 5 5 6
backwards: 5 6 5 -3 1
```

Reductio Ad Absurdum

```
my @up-to-four = 1, 2, 3, 4;
say “1 + 2 + 3 + 4 = “, sum @up-to-four;
say “Also = “, [+] @up-to-four;
Say “4! = “, [*] @up-to-four;
```

```
1 + 2 + 3 + 4 = 10
Also = 10
4! = 24
```

Higher-Order Functions

```
my @mixed-bag = 1, 2, -3, 6;
say “positives: “, grep { $^a > 0 }, @mixed-bag;
say “plus one: “, map { $^a + 1 }, @mixed-bag;
```

```
positives: 1 2 6
plus one: 2 3 -2 7
```

Queueing

```
my @queue = 1, 2, 3;
push @queue, 4;
say “first: “, @queue.first;
say “rest: “, @queue.rest;
```

```
first: 1
rest: 2 3 4
```

Recap

- Arrays are an ordered list of slots
- say “the first name is “, @name[0];
- @name[100] = “foo”;
- say @name[1,2];
- say “sorted: “, sort @name;
  - say “sorted: “, @name.sort;

Hashes

- %-variables are called “hashes”
  - Sometimes “dictionaries” or “associative arrays”
- Hashes are key-value stores
- Keys must be strings or integers
- Values can be anything
  - Including arrays, objects or even other hashes
- Built-in iterator methods
- Good for quick prototypes

Hash Basics

```
my %character =
    name => 'Dajeil',
    age => 197;
say %character{'name'}, ' is ',
%character{'age'};
```

```
Dajeil is 197
```

Digging deeper

```
my %eccleston =
    first => 'Christopher', last => 'Eccleston';
my %tennant =
    first => 'David', last => 'Tennant';
my @doctor =
    %eccleston, %tennant;
say “The first New Who Doctor is “, @doctor[0]{'first'};
```

The first New Who Doctor is Christopher

Hashes in the Balance

- Fast lookup – O(1)
- Self-documenting
  - Want a name? say %employee{'name'}
- Prone to typos
  - say %employee{'naem'}
  - Spend 10 minutes wondering where the name went
- Can insert anything into a value
  - %employee{'name'} = 'Demeisen';
  - %employee{'name'} = 1, 2, 7;

Enter the Object

- Compiled to real O(1) access
- Keys checked for typos
- Values checked for types
- Inherit from other “structs”
- A bit long-winded

Making the Transition

```
my %employee = name => 'Dajeil';
say “name: “, %employee{'name'};


class Employee { has $.name };
my Employee $employee =
    Employee.new( name => 'Dajeil' );
say “name: “, $employee.name;
```

High Finance in Perl 6

```
class Account {
    has $.name;
    has $.balance;
}
my $checking =
    Account.new(
        name => 'Bill', balance => 3200 );
say $checking.name, “ has €”, $checking.balance;

Bill has €3200
```

Being Methodical

```
class Account {
    has $.name;
    has $.balance;
    method debit( $amount ) {
        $!balance = $.balance + $amount;
    }
}
my $account = Account.new( name => 'Bill', balance => 32 );
$account.debit(100);
say $account.name, “ has a balance of €”, $account.balance;
```

Bill has a balance of €132

Ledger(demain)

```
my @bank =
    Account.new(...), Account.new(...);
for @bank -> $account {
    $account.add-interest( 0.1 );
    $account.audit();
}
...
```

Alice has €4400

Bob has €3300

The bank has €7700, and has paid €700 in interest.

My Answer part 1

```
class Account {
    has $.name;
    has $.balance;
    method audit() {
        say $.name, “ has €”, $.balance;
    }
    method add-interest( $amount ) {
        $!balance = $.balance + $balance * $amount;
    }
}
```

My answer part 2

```
my @bank =
    Account.new( name => 'Alice', balance => 4000 ),
    Account.new( name => 'Bob', balance => 3000 );
my $total = 0;
for @bank -> $account {
    $account.add-interest( 0.1 );
    $account.audit();
    $total = $total + $account.balance;
}
say “The bank has €“, $total;
```

Winding Things Up

- Variables are declared using 'my'
  - my $name;
- Use '$' to hold a single thing
- Use '@' or '%' to hold a list of things
  - @doctor[0] to peek at an entry
  - %doctor{'first-name'} to peek at an entry
- Use classes if you want to be strict or take actions.
- Containers can be used to create complex structures
- for @name -> $x { } iterates over arrays
- Match strings with ~~

Questions, or Bonus Round?

- The bonus round includes:
- Easier string manipulation
- Advanced OO techniques
- Advanced method calls

Bonus Slide: Strings

```
my $name = “Rose Tyler”;
say “My name is $name”;
```

```
My name is Rose Tyler
```

```
my $balance = 32;
say “After interest, the total is {$balance + 0.1 * $balance}”;
```

```
After interest, the total is 33.2
```

Bonus slide: Advanced OO

```
role Inventory {
    has Item @.item;
}
class Character {
    does Inventory;
    has Str $.name;
    has Int $.INT where 0 < * < 18;
}
```

Method signatures

```
method add-interest(
    Real $amount where 0 < * < 1 ) {
}
$account.add-interest(-1) # Fails at runtime
method set-type( Str $type = 'checking' ) {
    $.type = $type;
}
$account.set-type(); # sets to checking
$account.set-type('savings'); # sets to savings
```

Bonus slide: Advanced parsing

```
my $JSON = '{ “hello” : 1 }';
my regex quoted-string { \” .+? \” };
my regex integer { \d+ };
$JSON ~~ / '{' <quoted-string> ':' <integer> '}' /;
say “name is “, $/<quoted-string>;
say “and value is “, $/<integer>;
```