Quick Start: one-line “hello, world”

1. Create the file hello.chpl:
   ```chapel
   writeln("hello, world");
   ```
2. Compile and run it:
   ```bash
   $ chpl hello.chpl
   $ ./hello
   hello, world
   ```

Comments

// single-line comment
/* multi-line comment */
/* can be nested */
/* can be nested */

Primitive Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Default size</th>
<th>Other sizes</th>
<th>Default init</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>n/a</td>
<td>8, 16, 32</td>
<td>false</td>
</tr>
<tr>
<td>int</td>
<td>64</td>
<td>8, 16, 32</td>
<td>0</td>
</tr>
<tr>
<td>uint</td>
<td>64</td>
<td>8, 16, 32</td>
<td>0</td>
</tr>
<tr>
<td>real</td>
<td>64</td>
<td>32</td>
<td>0.0</td>
</tr>
<tr>
<td>imag</td>
<td>64</td>
<td>32</td>
<td>0.0i</td>
</tr>
<tr>
<td>complex</td>
<td>128</td>
<td>64</td>
<td>0.0+0.0i</td>
</tr>
<tr>
<td>string</td>
<td>n/a</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variables, Constants and Configuration

- `var x: real = 3.14;` - variable of type real set to 3.14
- `var isSet: bool;` - variable of type bool set to false
- `const epsilon: real = 0.01;` - runtime constant
- `param debug: bool = false;` - compile-time constant
- `config constant n: int = 100;` - compiled-in constant
- `config param d: int = 4;` - compiled-in constant

Compiles

```chapel
module M1 { var x = 10; }
module M2 {
  use M1;
  proc main() { writeln(x); }
}
```

Expression Precedence and Associativity

<table>
<thead>
<tr>
<th>Operators</th>
<th>Uses</th>
<th>Operators</th>
<th>Uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>- (left)</td>
<td>member access, call or index</td>
<td>** (right)</td>
<td>exponentiation</td>
</tr>
<tr>
<td>new (right)</td>
<td>creation of a new instance</td>
<td>:</td>
<td>cast</td>
</tr>
<tr>
<td>reduce scan</td>
<td>reduction, scan,</td>
<td>! ~ (right)</td>
<td>logical and bitwise negation</td>
</tr>
<tr>
<td>dmapped</td>
<td>apply domain map</td>
<td>* / %</td>
<td>multiplication, division, modulus</td>
</tr>
<tr>
<td>unary + - (right)</td>
<td>positive identity, negation</td>
<td>&amp;</td>
<td>bitwise/logical and</td>
</tr>
<tr>
<td>&lt;&lt; &gt;&gt;</td>
<td>shift left, shift right</td>
<td>^</td>
<td>bitwise/logical xor</td>
</tr>
<tr>
<td>&amp;</td>
<td>bitwise/logical or</td>
<td>+ -</td>
<td>addition, subtraction</td>
</tr>
<tr>
<td>...</td>
<td>range and open range construction</td>
<td>&lt;= =&gt; &lt; &gt;</td>
<td>ordered comparison</td>
</tr>
<tr>
<td>&lt;= =&gt; &lt; &gt;</td>
<td>equality comparison</td>
<td>== !=</td>
<td>short-circuiting logical and</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>short-circuiting logical or</td>
<td></td>
<td></td>
</tr>
<tr>
<td>by # align</td>
<td>used in loop headers</td>
<td>if</td>
<td>conditional expression,</td>
</tr>
<tr>
<td>in</td>
<td>for and foreach</td>
<td>for all and []</td>
<td>serial and order-indep. loop expr., parallel loop expressions</td>
</tr>
<tr>
<td>if</td>
<td>expression list</td>
<td>for</td>
<td></td>
</tr>
</tbody>
</table>

Casts and coercions

- `var i: 2.0:int;` - explicit conversion real to int
- `var x: real = 2;` - implicit conversion int to real

Conditional and Loop Expressions

- `var half = if i%2 then i/2+1 else i/2;` - conditional expression
- `writeln(for i in 1..n do i**2);` - serial and order-indep. loop expr., parallel loop expressions

Assignments

- Simple Assignment: `=`
- Compound Assignments: `+= -= *= /= %= *= -= |= ^= &= ||= <<= >>=`
- Swap Assignment: `<=`

Statements

```chapel
if cond then stmt1(); else stmt2();
if cond then stmt1() else stmt2();
select expr {
  when equiv1 do stmt1();
  when equiv2 do stmt2();
  otherwise stmt3();
}
do {...} while condition;
while condition {...} single-statement forms:
for index in iterable {...} do stmt();
foreach index in iterable {...}
try {...} catch error {...} label outer for ...
break; or break outer;
continue; or continue outer;
```

Procedures

```chapel
proc bar(r: real, i: imag): complex {
  return r + i;
}
proc foo(i) do return i**2 + i + 1;
```

Formal Argument Intents

<table>
<thead>
<tr>
<th>Intent</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>copy-initialized in</td>
</tr>
<tr>
<td>out</td>
<td>copied out</td>
</tr>
<tr>
<td>inout</td>
<td>copied in and out</td>
</tr>
<tr>
<td>ref</td>
<td>passed by reference</td>
</tr>
<tr>
<td>const in</td>
<td>copied in; local modifications are disallowed</td>
</tr>
<tr>
<td>const ref</td>
<td>passed by reference; local modifications are disallowed</td>
</tr>
<tr>
<td>const</td>
<td>passed by value or by reference; local and caller modifications are disallowed</td>
</tr>
<tr>
<td>default</td>
<td>like const for most types; like ref for syncs and atomics</td>
</tr>
</tbody>
</table>

Named Formal Arguments

```chapel
proc foo(arg1: int, arg2: real) {...} foo(arg2=3.14, arg1=2);
```

Default Values for Formal Arguments

```chapel
proc foo(arg1: int, arg2: real = 3.14); foo(2);
```
Records

```plaintext
record Point {
  var x, y: real;
}
var p: Point;
```

Classes

```plaintext
class Circle {
  var p: Point;
  real r: real;
}
```

Unions

```plaintext
union U {
  var i: int;
  var r: real;
}
```

Tuples

```plaintext
var pair: (string, real);
var coord: 2*int;
coord = ("one", 2.0);
var (s, r) = pair;
```

Enumerated Types

```plaintext
enum day {sun, mon, tue, wed, thu, fri, sat};
var today: day = day.fri;
```

Ranges

```plaintext
var every: range = 0..n;
var evens = every by 2;
var R = evens # 5;
var odds = evens align 1;
var open = 0..<cn;
```

Domains and Arrays

```plaintext
var rectangular: domain(1);  1-d domain (index set)
const D = (1..n);  domain literal
var A: [D] real;  array of real numbers
var Set: domain(int);  associative domain
Set += 3;  add index to domain
var SD: sparse subdomain(D);  sparse domain
```

Domain Maps

```plaintext
use BlockDist;
const D = (1..n) dmapped distrib. domain w/ block
blockDist(boundingBox=(1..n));  distribution
var A: [D] real;
```

Data Parallelism and Task Intents

```plaintext
forall i in D do A[i] = 1.0;  domain iteration
forall a in A do a = 1.0;  array iteration
forall a in A a = 1.0;  promoted addition and array assignment
forall a in A task intents: [const in, [const ref] reduce
with [const in factor, + reduce sum]
do sum reduce= a * factor;
```

Reductions and Scans

```plaintext
var sum = + reduce A;  123 => 6
var pre = + scan A;  123 => 136
var ml= minloc reduce (A, A.domain);
```

Iterators

```plaintext
iter squares(n: int) {  serial iterator
  for i in 1..n do
    yield i**2;
}
for s in squares(n) do ...  loop over iterator
```

Zipper Iteration

```plaintext
for (i, s) in zip(1..n, squares(n)) do ...
```

Extern Declarations

```plaintext
extern proc C_function(x: int);
extern "C_name" var C_variable: real;
extern { /* C code here */ }
```

Task Parallelism

```plaintext
begin task();
obegin { task1(); task2(); }
coball ip i in iterable do task(i);
sync { begin task1(); begin task2(); }
serial condition do stmt();
```

Atomic Example

```plaintext
var count: atomic int;
if count.fetchAdd(1)==n-1 then
  done = true;  //th task to arrive
```

Synchronization Examples

```plaintext
var data: sync int;
data.writeEF(produce1());
consume(data.readFE());
data.writeEF(produce2());
consume(data.readFE());
```

Locality

```plaintext
config const numLocales: int; $./prog -nl 4
const LocaleSpace = {0..numLocales-1};
const Locales: [LocaleSpace] locale;
```

Example

```plaintext
var c: owned Circle;
on Locales[i] {  migrate task to new locale
  writeln(here);  print the current locale
  c = new Circle();  allocate class on locale
  writeln(c.locale);  query locale of class instance
} on c do { ... }  data-driven task migration
```

User Resources

```plaintext
https://chapel-lang.org/users.html
```