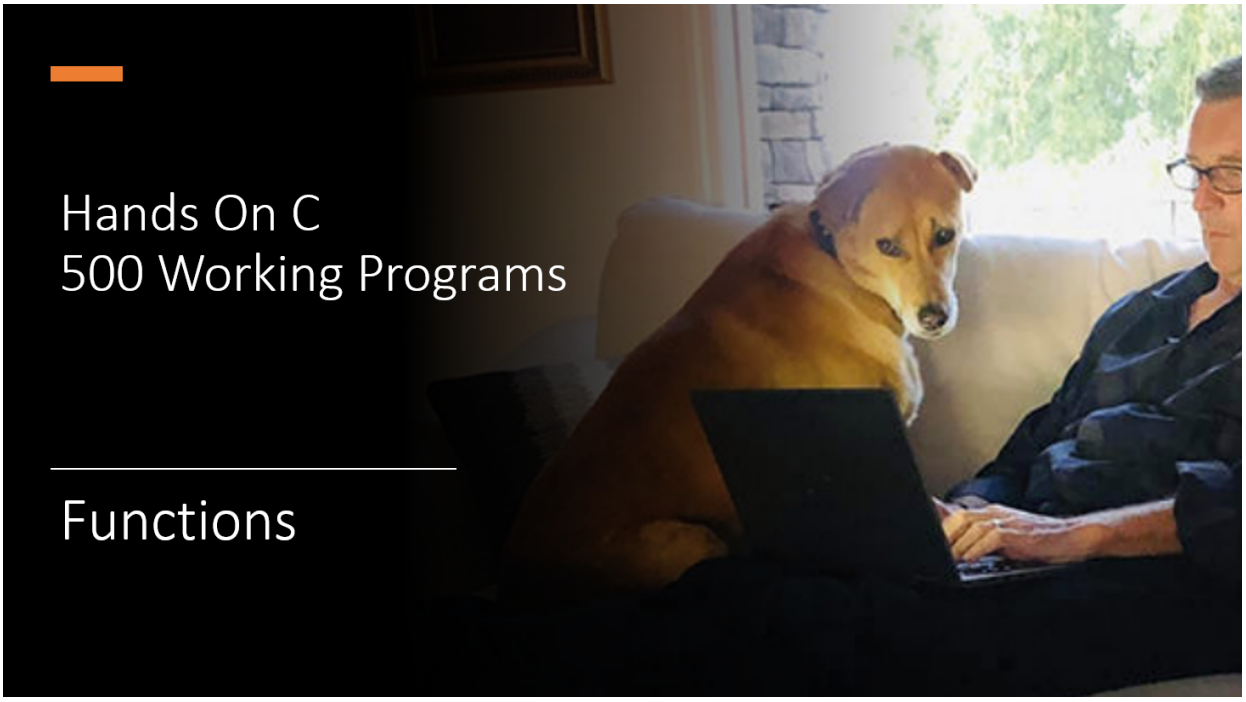

Understanding Functions

In [1]:
```c
#include <stdio.h>

int main(void)  // Main is a function
{
   printf("Hello, World\n");  // printf is a function
}
```

```
Hello, World
```

```
Functions

1. Have a unique name
2. Return a type or void
3. Receive one or more arguments called parameters
4. Execute when they are called
```

In [2]:
```c
#include <stdio.h>

void aboutMe(void)
{
    printf("Name: Kris Jamsa\n");
    printf("Home Town: Prescott, Arizona\n");
    printf("Hobbies: coding\n");
}

int main(void)  // Main is a function
{
    aboutMe();
}
```

```
Name: Kris Jamsa
Home Town: Prescott, Arizona
Hobbies: coding
```

In [3]:
```c
#include <stdio.h>

void aboutMe(void)
{
    printf("Name: Kris Jamsa\n");
    printf("Home Town: Prescott, Arizona\n");
    printf("Hobbies: coding\n");
}

int main(void)  // Main is a function
{
    printf("Programs start in main\n");
    aboutMe();
    printf("Programs should end in main\n");
}
```

```
Programs start in main
Name: Kris Jamsa
Home Town: Prescott, Arizona
Hobbies: coding
Programs should end in main
```

Programs Pass Data to Functions Using Parameters

In [4]:
```c
#include <stdio.h>

int main(void)
{
    printf("Hello\n");                          // one parameter
    printf("Age %d\n", 50);                     // two parameters
    printf("Age %d %6.2f", 50, 100000.0);       // three parameters
}
```

```
Hello
Age 50
Age 50 100000.00
```

Creating a Function that Receives a Parameter

In [5]:
```c
#include <stdio.h>

void sayHello(int count)
{
    for (int i = 0; i < count; i++)
      printf("Hello, ");

    printf("World\n");
}

int main(void)
{
    sayHello(1);
    sayHello(2);
    sayHello(3);
}
```

```
Hello, World
Hello, Hello, World
Hello, Hello, Hello, World
```

Declaring Variables within a Function

In [6]:
```c
#include <stdio.h>

int max(int a, int b, int c)
{
    int max = a;    // assume a is the max

    if (b > max)
      max = b;

    if (c > max)
      max = c;

    return(max);
}

int main(void)
{
    int a = 1, b = 2, c = 3;

    printf("Max of %d %d and %d is %d\n", a, b, c, max(a, b, c));
}
```

```
Max of 1 2 and 3 is 3
```

In [7]:
```c
#include <stdio.h>

int max(int a, int b, int c)
{
    int max = a;    // assume a is the max

    if (b > max)
      max = b;

    if (c > max)
      max = c;

    return(max);
}

float floatmax(float a, float b, float c)
{
  return((a > b) ? ((a > c) ? a : c) : ((b > c) ? b : c));
}

int main(void)
{
    int a = 1, b = 2, c = 3;
    float d = 1.1, e = 3.3, f = 2.2;

    printf("Max of %d %d and %d is %d\n", a, b, c, max(a, b, c));
    printf("Max of %3.1f %3.1f and %3.1f is %3.1f\n", d, e, f, floatmax(d, e, f))
}
```

```
Max of 1 2 and 3 is 3
Max of 1.1 3.3 and 2.2 is 3.3
```

Revisiting main's Return Value

In [8]:
```c
#include <stdio.h>

int main(void)
{
    printf("Hello, World\n");
}
```

```
Hello, World
```

In [9]:
```c
#include <stdio.h>

int main(void)
{
    printf("Hello, World\n");
    return(1);
}
```

```
Hello, World

[C kernel] Executable exited with code 1
```

In [10]:
```c
#include <stdio.h>

void main(void)
{
    printf("Hello, World\n");
}
```

```
Hello, World

[C kernel] Executable exited with code 13
```

Understanding main's Parameters

```
Command Line

C:\> Copy Filename.txt Filename.bak   <Enter>
```

In [11]:
```
#include <stdio.h>

int main(int argc, char *argv[])
{
   printf("argc %d\n", argc);

   for (int i = 0; i < argc; ++i)
     printf("argv[%d] is %s\n", i, argv[i]);
}
```

```
argc 1
argv[0] is /tmp/tmp9ufi0shw.out
```

For Readability, Functions Should Only Have One Return Statement

In [12]:
```c
#include <stdio.h>

int compare(int a, int b)
{
    if (a == b)
      return(0);
    if (a > b)
      return(1);
    else // b > a
      return(2);
}

int main(void)
{
    printf("Comparing %d and %d yields %d\n", 33, 22, compare(33, 22));
}
```

```
Comparing 33 and 22 yields 1
```

In [13]:
```c
#include <stdio.h>

int compare(int a, int b)
{
    int result;

    if (a == b)
      result = 0;
    if (a > b)
      result = 1;
    else // b > a
      result = 2;

    return(result);
}

int main(void)
{
    printf("Comparing %d and %d yields %d\n", 33, 22, compare(33, 22));
}
```

```
Comparing 33 and 22 yields 1
```

Understanding Function Prototypes

In [14]:
```
#include <stdio.h>

int main(void)
{
    printf("Comparing %d and %d yields %d\n", 33, 22, compare(33, 22));
}

int compare(int a, int b)
{
    if (a == b)
      return(0);
    if (a > b)
      return(1);
    else // b > a
      return(2);
}
```

```
/tmp/tmpsqjquf7b.c: In function ‘main’:
/tmp/tmpsqjquf7b.c:5:55: warning: implicit declaration of function ‘compare’ [-
Wimplicit-function-declaration]
     printf("Comparing %d and %d yields %d\n", 33, 22, compare(33, 22));
                                                       ^~~~~~~

Comparing 33 and 22 yields 1
```

In [15]:
```c
#include <stdio.h>

int compare(int, int);  // Function prototype

int main(void)
{
    printf("Comparing %d and %d yields %d\n", 33, 22, compare(33, 22));
}

int compare(int a, int b)
{
    if (a == b)
      return(0);
    if (a > b)
      return(1);
    else // b > a
      return(2);
}
```

```
Comparing 33 and 22 yields 1
```

Understanding Function Overhead

In [*]:
```c
#include <stdio.h>
#include <time.h>

float add_em(long int a, float b)
 {
    float result;
    result = a + b;
    return(result);
 }

 int main(void)
  {
    float result = 0;
    time_t start_time, stop_time;

    printf("Calculating...\n");
    time(&start_time);

    for (long int i = 1; i <= 1000000000L; i++) // 1,000,000,000 numbers
      result += i;

    time(&stop_time);

    printf("Using loop %ld seconds\n", stop_time - start_time);

    printf("Calculating...\n");
    time(&start_time);

    for (long int i = 1; i <= 1000000000L; i++)
      result = add_em(i, result);

    time(&stop_time);

    printf("Using function %ld seconds\n", stop_time - start_time);
 }
```

```
Calculating...
```

Understanding Global Variables

In [*]:
```c
#include <stdio.h>

int a, b, c;   // global variables

void initialize(void)
{
    a = 1000;
    b = 2000;
    c = 3000;
}

int main(void)
{
    initialize();
    printf("%d %d %d\n", a, b, c);
}
```

Local and Global Variable Name Conflicts

In [*]:
```c
#include <stdio.h>

int a = 1000, b = 2000, c = 3000;   // global variables

void hasConflict(void)
{
   int a = 1, b = 2, c = 3;

   printf("In hasConflict %d %d %d\n", a, b, c);
}

int main(void)
{
    hasConflict();
    printf("In main %d %d %d\n", a, b, c);
}
```

Understanding Call by Value

In [*]:
```c
#include <stdio.h>

void change_values(int a, int b)
{
    a = 1000;
    b = 2000;
}

int main(void)
{
    int a = 1, b = 2;

    change_values(a, b);
    printf("Ending Values %d %d\n", a, b);
}
```

Understanding Call by Reference

In [*]:
```c
#include <stdio.h>

void change_values(int *a, int *b)
{
    *a = 1000;
    *b = 2000;
}

int main(void)
{
    int a = 1, b = 2;

    change_values(&a, &b);
    printf("Ending Values %d %d\n", a, b);
}
```

In [*]:
```c
#include <stdio.h>

void swapValues(float *a, float *b)
{
    float temp = *a;
    *a = *b;
    *b = temp;
}

int main(void)
{
     float big = 1000.0, little = 0.0;

     swapValues(&big, &little);
     printf("Big %f Little %f\n", big, little);
}
```

C Passes Arrays to Functions by Reference (Address)

In [*]:
```c
#include <stdio.h>

void showArray(int array[], int numberOfElements)
{
    for (int i = 0; i < numberOfElements; ++i)
      printf("%d\n", array[i]);
}

int main(void)
{
   int values[5] = {10, 20, 30, 40, 50 };
   showArray(values, 5);
}
```

In [*]:
```c
#include <stdio.h>
#include <ctype.h>

void stringToUppercase(char string[])
{
   for (int i = 0; string[i]; ++i)
     string[i] = toupper(string[i]);
}

int main(void)
{
   char string[] = "abcdef";

   stringToUppercase(string);
   printf("%s\n", string);
}
```

Static Variables Remember There Previous Value

In [*]:
```
#include <stdio.h>

long int totalCharacters(char string[])
{
  int static count = 0;

  for (int i = 0; string[i]; ++i)
    count++;

  return(count);
}

int main(void)
{
   printf("Characters counted so far %ld\n", totalCharacters("ABCDEF"));
   printf("Characters counted so far %ld\n", totalCharacters("GHIJKL"));
   printf("Characters counted so far %ld\n", totalCharacters("MNOPQR"));
   printf("Characters counted so far %ld\n", totalCharacters("STUVWXYZ"));
}
```

Understanding the const Qualifier

In [*]:
```
#include <stdio.h>

int main(void)
{
   const int i = 1001;

   for (i = 0; i < 1000; ++i)
     printf("%d\n", i);
}
```

Understanding Recursion

```
Value     Calculation      Result        Factorial
1         1                1             1
2         2*1              2             2*Factorial(1)
3         3*2*1            6             3*Factorial(2)
4         4*3*2*1          24            4*Factorial(3)
5         5*4*3*2*1        120           5*Factorial(4)
```

In [*]:
```c
#include <stdio.h>

int factorial(int n)
{
   if (n == 1)
     return(1);
   else
     return(n*factorial(n-1));
}

int main(void)
{
  for (int i = 1; i <= 5; i++)
    printf("Value %d factorial(%d) is %d\n", i, i, factorial(i));
}
```

In [*]:
```c
#include <stdio.h>

int factorial(int value)
 {
   printf("In factorial with the value %d\n", value);

   if (value == 1)
    {
      printf("Returning the value 1\n");
      return(1);
    }
   else
    {
      printf("Returning %d * factorial(%d)\n",
         value, value-1);
      return(value * factorial(value-1));
    }
 }

int main(void)
 {
   printf("The factorial of 4 is %d\n", factorial(4));
 }
```

In [*]:
```c
#include <stdio.h>

void displayInReverse(char *string)
{
  if (*string != '\0')
    {
      displayInReverse(string+1);  // Next character
      printf("%c", *string);
    }
}

int main(void)
{
   char string[] = "Hello, world!";

   displayInReverse(string);
}
```

Determining When to Use Recursion

In [*]:
```c
#include <stdio.h>
#include <time.h>

int string_length(const char *str)
 {
   int length = 0;

   while (*str++)
     length++;

   return(length);
 }

int main(void)
 {
   long int counter;

   time_t start_time, end_time;

   time(&start_time);

   for (counter = 0; counter < 100000000L; counter++)
     string_length("Hello, World");

   time(&end_time);

   printf("Processing time %ld second\n", end_time - start_time);
 }
```

In [*]:
```c
#include <stdio.h>
#include <time.h>

int string_length(const char *str)
 {
   if (*str)
     return(1 + string_length(str+1));
   else
     return(0);
 }

int main(void)
 {
   long int counter;

   time_t start_time, end_time;

   time(&start_time);

   for (counter = 0; counter < 100000000L; counter++)
     string_length("Hello, world");

   time(&end_time);

   printf("Processing time %ld seconds\n", end_time - start_time);
 }
```

Creating a Function that Supports a Varying Number of Parameters

```
printf("Hello, world");

printf("Hello, world %d\n", 1000);

printf("Hello, world %d\n", 1000, 2000);
```

In [*]:
```
#include <stdio.h>
#include <stdarg.h>

int add_values(int value, ...)
 {
    va_list argument_ptr;
    int result = 0;

    if (value != 0)
      {
        result += value;
        va_start(argument_ptr, value);

        while ((value = va_arg(argument_ptr, int)) != 0)
          result += value;

        va_end(argument_ptr);
      }
    return(result);
 }

int main(void)
 {
   printf("Sum of 3 is %d\n", add_values(3, 0));
   printf("Sum of 3 + 5 is %d\n", add_values(3, 5, 0));
   printf("Sum of 3 + 5 + 8 is %d\n", add_values(3, 5, 8, 0));
   printf("Sum of 3 + 5 + 8 + 9 is %d\n", add_values(3, 5, 8 , 9, 0));
 }
```

Creating a Function that Supports Varying Parameter with Different Types

In [*]:
```
#include <stdio.h>
#include <stdarg.h>

double add_values(char *str, ...)
{
    va_list marker;

    double result = 0.0;

    va_start(marker, str);  // mark first additional argument

    while (*str)            // examine each character in the string
      {
      if (*str == '%')        // if not a %_ format specifier, skip it
       {
         switch (*(++str)) {
           case 'd': result += va_arg(marker, int);
                break;
           case 'f': result += va_arg(marker, double);
           break;
         }
      }
     str++;
    }

   va_end(marker);
   return(result);
 }


int main(void)
 {
   double result;

   printf("Result %f\n", add_values("%f", 3.3));
   printf("Result %f\n", add_values("%f %f", 1.1, 2.2));
   printf("Result %f\n", add_values("%f %d %f", 1.1, 1, 2.2));
   printf("Result %f\n", add_values("%f %d %f %d", 1.1, 1, 2.2, 3));
 }
```

What You will Learn Next

```
To save programmers time, C compilers provide a library of built-in functions
your programs can call.

printf("The absolute value of -1 is %d\n", abs(-1));
```