Quiz 1: Functions and Procedures

Outline

- Basics
- Control Flow
- While Loops
- Expressions and Statements
- Functions

Primitive Data Types

3 simple data types: number, string, boolean

Numbers store numerical data

Strings store textual data

Booleans store true or false

Primitives Examples

What are the types of these variables?

```
let c: _____ = 5;
let o: _____ = "UNC";
let m: _____ = "true";
let p: _____ = false;
```

Declaring a Variable

Springing a variable into existence

Syntax:

```
let variableName: variableType;
```

Initializing a Variable

After declaring a variable, we can assign it a value.

Syntax:

```
let staplesButton: string;
staplesButton = "That was easy";
```

Declaring and Initializing… AT THE SAME TIME

Very simple to declare and initialize in the same line, simply combine the two statements.

Very common to see this when you know an initial value for a variable.

```
let betterStaplesButton: string = "That was easier";
```

Type Inference

Sometimes you explicitly declare a type, sometimes you don’t

Programs are able to **infer** the type of the assignment

Conditional Operators: Equality

Equality Operators

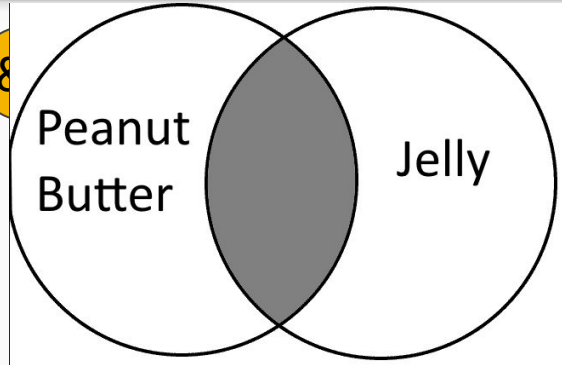
- ===
  - **equal to** operator -- THREE equals symbols in a row.
- !==
  - **not equal to** operator -- ! symbol followed by TWO equals symbols
    i. The ! means “NOT”

Conditional Operators: Relational

Relational Operators

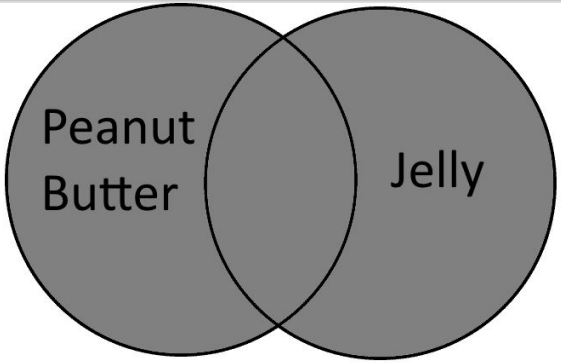
- **>**
  - Greater than
- **>=**
  - Greater than or equal to
  - “At least”
- **<**
  - Less than
- **<=**
  - Less than or equal to
  - “At most”

Conditional Operators: Logical && Negation



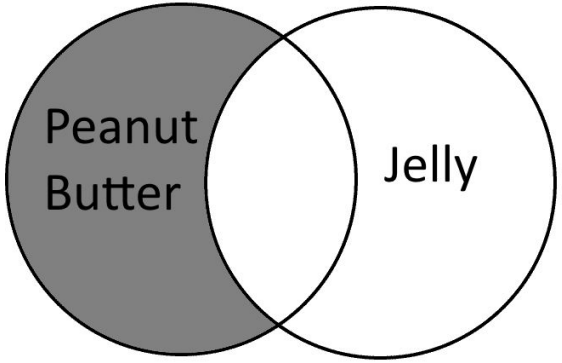
AND

Using AND, this search would only retrieve results with Peanut Butter and Jelly.



OR

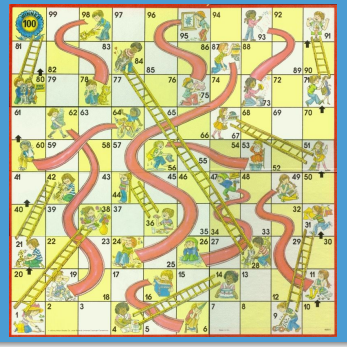
Using OR, this search would retrieve results with peanut butter, with jelly, and with both.

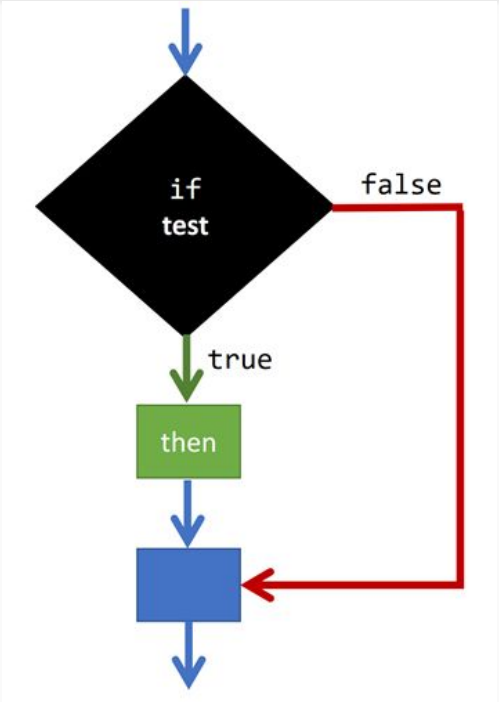


NOT

Using NOT, this search would retrieve results with peanut butter, and exclude those with jelly or PB with jelly.

Control Flow

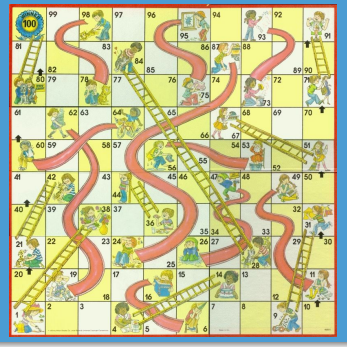


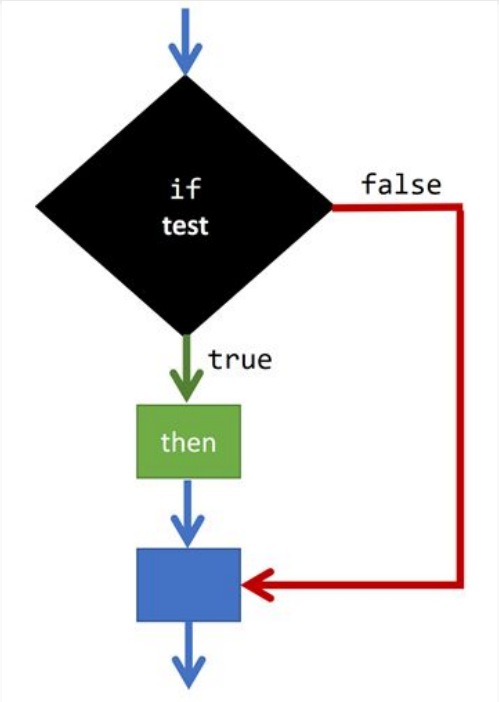


Order in which statements, instructions, and function calls are executed

- Path of a program changes depending on the values of variables during execution
- Generally, the computer reads code as we read books… BUT it depends on
  - If-then-else
  - Function calls
  - Loops

Control Flow





- Variables : Program :: Characters : Book
  - Declare variables before using them in your code
- If-then-else
- Function calls
- While loops

If-then and if-then-else

```
if (<conditional statement>) {
    // code to execute ONLY if conditional statement is true
} else {
    // code that executes ONLY when conditional statement is false
}
```

- Use boolean expressions (evaluating to true or false)
- Useful when you want for the program to do different things based on a variable’s value

If-then and if-then-else

Say we're making someone's favorite food. The food that we make will depend on the recipient!

What will this program print out?

What's my favorite food?

pretzels

```
import { print } form "introcs";

let person = "Stanley";

print("What's my favorite food?");

if (person === "Kevin") {
  print("chili");
} else {
  if (person === "Stanley") {
    print("pretzels");
  } else {
    print("cake");
  }
}
```

If-then and if-then-else

Say we're making someone's favorite food. The food that we make will depend on the recipient!

What will this program print out if person is now "Meredith"?

What's my favorite food?

cake

```
import { print } form "introcs";

let person = "Meredith";

print("What's my favorite food?");

if (person === "Kevin") {
  print("chili");
} else {
  if (person === "Stanley") {
    print("pretzels");
  } else {
    print("cake");
  }
}
```

How can we simplify this code?

Avoid the need for a nested if statement by using an else if!

```
import { print } form "introcs";

let person = "Stanley";

print("What's my favorite food?");
if (person === "Kevin") {
    print("chili");
} else {
    if (person === "Stanley") {
        print("pretzels");
    } else {
        print("cake");
    }
}
```

nesting

if (<conditional statement>) {
  // executes if true
} else if (<conditional statement>) {
  // executes if true and first conditional statement is false
} else {
  // executes if both conditional statements were false
}

Implementing an else if

Avoid the need for a nested if statement by using an else if!

1. Remove the opening curly brace after the else, and remove the matching closing curly brace
2. Move the nested **if** to be right next to the **else** keyword, and fix the indentation and spacing

- More clean, concise, speedy, easy to read

```
import { print } form "introcs";

let person = "Stanley";

print("What's my favorite food?");

if (person === "Kevin") {
  print("chili");
} else if (person === "Stanley") {
    print("pretzels");
} else {
    print("cake");
}
```

Unreachable Statements

WS0 Review:
Is any of this code unreachable?

When a statement can't possibly execute in any circumstance
Common causes:

- Nested if statements within an else
- Code after a return statement (when return is not in an if block, etc.)

```
let getName = (y: number): string => {
    if (isEven(y)) {
        return "Michael";
    } else {
        if (y === 4) {
            return "Dwight";
        } else {
            if (y === 3) {
                return "Jim";
            }
            return "Stanley";
        }
    }
};
```

While Loops

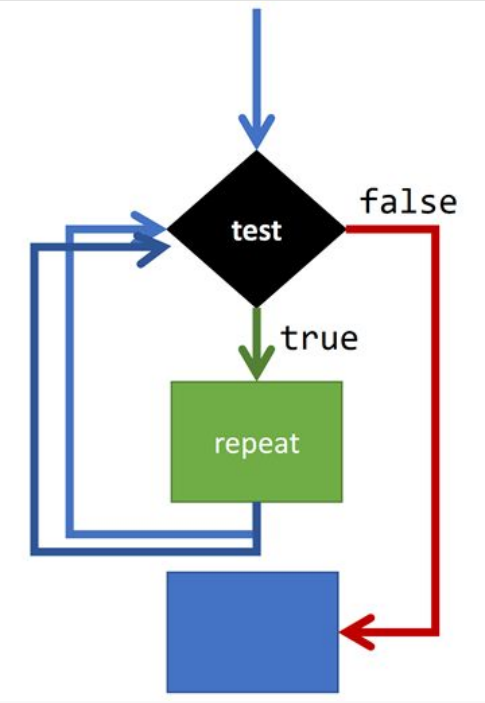
While (<boolean expression “test”>) {

// repeat block - statements in braces run when the test is true

}

- Code runs as long as the boolean expression is true
- After the last statement in the repeat block completes, computer jumps back to the test
- Repeats code **while** boolean expression continues to be true after executing the repeat block

While Loops: Control Flow



When to use?

- Repeating a task a specific number times

Avoiding infinite loops:

- Make sure that your boolean expression will evaluate to false eventually!!
  - Increment your counter, change variable from true to false, etc.

While Loops vs. If-then-else

WHILE

- Computer jumps back up to the test after executing the repeat block

BOTH

- Test in parentheses must be a boolean expression
- If test evaluates to true: computer moves to the first line of code in the following { }
- If test evaluates to false, computer will *jump over* the following { }

IF

- No repetition
- Opportunity to check for different boolean expressions with else if or execute default code with else

Check-in and Hot Date

Check-in code: **A4043**

Find a partner, introduce yourselves to each other!

Talk about why you are taking COMP110, and what you did last summer!

Expressions and statements

A **statement** is an instruction that you are giving to the computer.

Expressions are statements made up of variables, literals, operators, or functions that can be evaluated to produce a single result.

Example Expressions

What type do the expressions on the right-hand side of the assignment operator evaluate to?

What does the last line evaluate to?

```
let s = "My favorite number is " + 5;
let funNum = 5 / 5 * 5 + 5 - 5;
let x: string = someFunctionThatReturnsAString();
print("print it");
```

Functions

A **function** is a named section of a program that performs a specific procedure.

You can use functions to do ~anything~

Defining Functions

Parts of function definitions:

- Name: the name of our function
- Parameters: placeholder variables for expected inputs
- Return type: the type of value we want to return from the function
- Body: the statements or lines of code that make up the function
  - This will often include return statements

We define functions using the following syntax:

```
let <name> = (<parameters>): <return type> => {
    //function body
};
```

```
let applePie = (one:string, two:string) : string => {
  return "We combined " + one + " with " + two + "and now we have apple pie yummm";
};
```

Calling Functions

- To run the code in our functions, we call them
- If the function definition has parameters, we must pass values (arguments) into the function call
- Arguments must match parameters in type and order
- To call functions we use the following syntax:

```
<name>(<arguments>);
```

Calling Functions

Given the function definition below, how would you call this function so that we combine **apples** with **sugar** to make our apple pie?

```
let applePie = (one:string, two:string) : string => {
  return "We combined " + one + " with " + two + "and now we have apple pie yummm";
};
```

```
applePie("Apples", "Sugar");
```

Arguments vs Parameters

Parameters are the **pieces** of information that a function needs to run.

Arguments are the **actual** values provided to a function.

```
let applePie = (one:string, two:string) : string => {
  return "We combined " + one + " with " + two + "and now we have apple pie yummm";
};
```

```
applePie("Apples", "Sugar");
```

Return Statements

Return statements stop execution of a function, and return a single value back to the location where it was called.

Execution stops even if we are within an **if** statement, **while** loop, etc.

Return value must match the return type of a function.

Practice 1

Given

```
let a = 5;
let b = 12;
```

which answer choice swaps these 2 values correctly?

a)

```
a = b;
b = a;
```

b)

```
let temp = a;
b = temp;
a = b;
```

c)

```
let temp = a;
a = b;
b = temp;
```

d)

```
let temp = a;
a = b;
b = temp;
```

Practice 2

Create function calls that can result in each letter being returned.

Are all of the branches reachable?

```
let booleanFun = (be: number): string => {
  if (be < 10) {
    return "A";
  } else {
    if (be > 10) {
      return "B";
    } else {
      if (be > 100) {
        return "C";
      } else {
        return "D";
      }
    }
  }
}
```

Practice 3

If you had to rename this function, what would you call it?

It’s the power function!

```
let kajow = (a: number, b: number): number => {
  if (b <= 0) {
    return 1;
  } else if (b === 1) {
    return a;
  } else {
    let c = a;
    while (b > 1) {
      c = c * a;
      b--;
    }
    return c;
  }
}

kajow(4, 3);
```