

スクーリング指導書

令和2年度「専修学校リカレント教育総合推進プロジェクト」

スクーリング指導書

技術者学び直し講座のモデルとなるITエンジニアを対象とした
eラーニング講座開設およびガイドラインの実証

目次

スクーリング指導書.....	04
資料	
Ruby講義資料1	18
Ruby講義資料2.....	39
Ruby演習問題_出題1	64
Ruby演習問題_出題2	70
Ruby演習問題_解答1	77
Ruby演習問題_解答2	84
e-learning 事前準備手順	92
Ruby on Rails 講義.....	104
アジャイルソフトウェア開発確認問題	112

本書は、eラーニングにおいてスクーリングを行う際の指導上の注意点等を整理したものである。2020年初から始まった新型コロナ感染流行を受けて、リアルでの集合型の授業実施が難しくなったことから、主にリモートで行うスクーリングについて記述している。

1. スクーリングについて

【スクーリング目的】

スクーリングは、eラーニングの受講完走度の向上、参加者交流による学習意欲や仲間意識の促進、より深い学習、学習内容の理解度確認等を目的に行われる。詳しくは、本研究事業で作成したガイドラインに記載している。

【スクーリングの実施タイミング】

1) 開講時

受講開始の意識付けのために、開始時のオリエンテーションはぜひともスクーリングで行って欲しい。特に、実習・演習を伴うeラーニングの場合は、実習環境の構築等につまずくと全く受講のステップに進めなくなるので、必ずスクーリングの中で構築作業を行うべきである。構築した環境で実際に実習・演習を行ってみて、実習環境の使用方法的な理解まで確認することが重要である。

2) 知識・技術レベルのステップの移行時

知識や技術を習得していく段階で、必ず理解しておかなければならないポイントがある。そのポイントの理解度のチェックと受講者の進度合わせのために、実習・演習を中心に、グループ作業等をスクーリング行うことは有効である。

3) より深い理解の獲得と習得度評価（修了時）

学習したことに対する自信獲得と以降の継続学習のモチベーション保持のために、最終課題等の実習・演習のグループ作業をスクーリングで行うことは有効である。この過程で、受講者の習得度評価も可能になる。

【スクーリングの実施上の注意】

学習は、個人よりも集団の中で、お互いに影響しあいながら行うことが効果的である。eラーニングにおいては、一堂に顔を合わす機会はスクーリングしかないため、スクーリングは貴重な機会になる。特に、オンラインでスクーリングを行う場合は、その顔合わせであることを意識して、できるだけ学習者同志の交流ができるようにすることが重要である。スクーリングの最初では、講師およびサポートメンバー、受講者全員が、出席確認および、受講環境、映像と音声の確認を兼ねて、簡単な自己紹介を行うことを勧める。

<Zoom 上での事項紹介画面>



【事前準備】

オンライン開催は、機器やオンラインアプリ、通信状況等のトラブルで、シラバス通りの進行に遅延を発生させないことが重要である。受講者のパソコンやインターネット環境の設定は、それぞれのパソコンの状況の違いから、事前に通知して個別対応をしてもらうとともに、スクーリングの開始時には確認およびフォローが同時に行える体制を準備する必要がある。そのためには、

- ・メイン講師：1名
- ・サポートスタッフ：受講者5名に1担当を想定して、スタッフを確保する。

2. オンラインでのスクーリングについて

オンラインでスクーリングを行うためには、オンライン会議用アプリ等の便利なコミュニケーションツールを使用することが多い。オンライン会議アプリを使用することにより、

- ・オンライン会議アプリを会社や自宅のパソコン等情報機器に導入すれば、ミーティングの参加者全員で予定を調整して同じ場所に集まる必要がない。
- ・ビデオ・テレビ会議ツールのように通話に必要な専用機器や通信工事が不要となる。
- ・基本的には手持ちのパソコンやスマホなどの端末があれば、システムを利用できる。

- ・参加者は会議の開催時刻までにオンライン会議アプリを開けば、ミーティングへの参加が可能となる。
- ・講師や支援スタッフは web 上で資料を共有するだけですむ。

【オンライン会議アプリ】

オンライン上での一斉授業、個別フォロー授業で利用する。

1) Zoom



(Zoom の簡単な紹介)

クラウドコンピューティングを使用した Web 会議サービスで、ミーティングルームを開設し、ミーティング ID やパスワードを共有するユーザー同士で多地点と同時に Web 会議を行うことができる。複雑な設定無しに、一般的なファイアウォールや [NAT](#) 内からでも通信が可能で、ミーティング中に挙手やチャット送信をすることができる。部屋を映したくない場合バーチャル背景を使い、画像を背景として使用できる機能もある。参加者をグループ分けしてサブ会議室を設定することもできる。(Wikipedia)

2) Discord

(Discord の簡単な紹介)

ボイスチャットおよびテキストチャットの両機能を兼ね備えており、Zoom などのオンライン会議ツールと、Slack などのチャットツールを合わせたようなツールである。1 つのボイスチャンネルに参加しながら、複数のテキストチャンネルを切り替えて閲覧できる。ボイスチャンネルごとにチャンネルの参加者が表示されており、目的のチャンネルをさがしやすく、チャンネルの移動も 1 クリックで容易にできる。



【コミュニケーションツール】

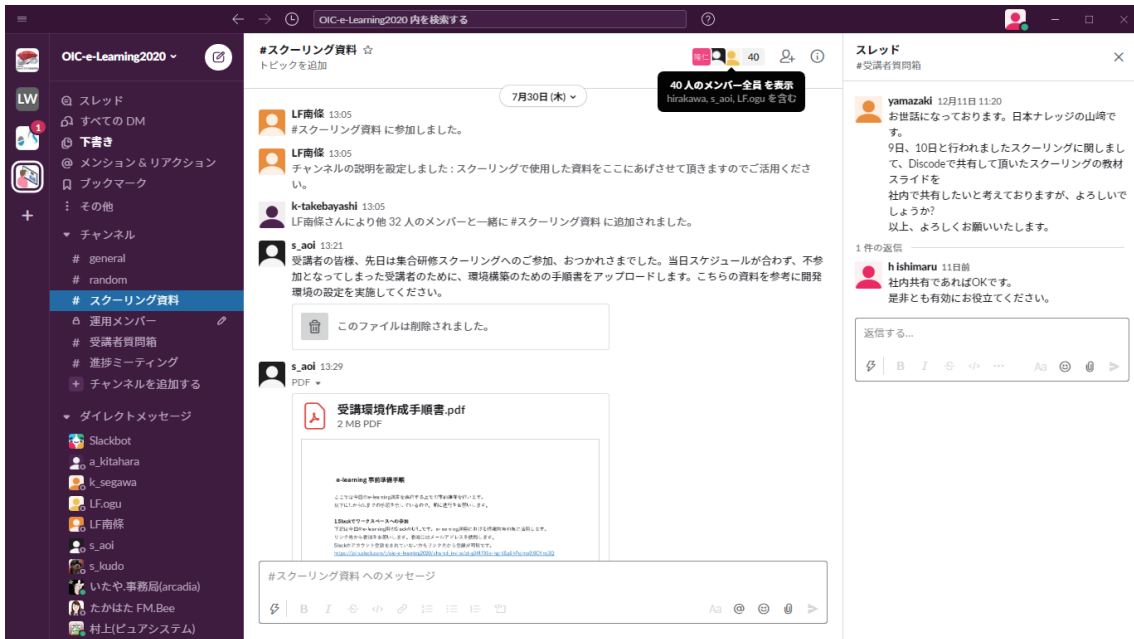
e-Learning 受講期間中のコミュニケーションのツールとして使用する。

1) Slack

(Slack の簡単な説明)

グループチャット、1対1のメッセージング (Direct Message)、音声通話を Web サービスとして提供している。また同様の機能をデスクトップアプリ (Windows、macOS) およびスマートフォンアプリ (iOS、Android) でも提供している。Google ドキュメント、Dropbox、Heroku、Crashlytics、GitHub、PagerDuty、Zendesk などを含む各種サードパーティーのサービスと連携することが出来るようになっており、Slack 内部のすべてのコンテンツは、一つの検索ボックスから検索できるようになっている。

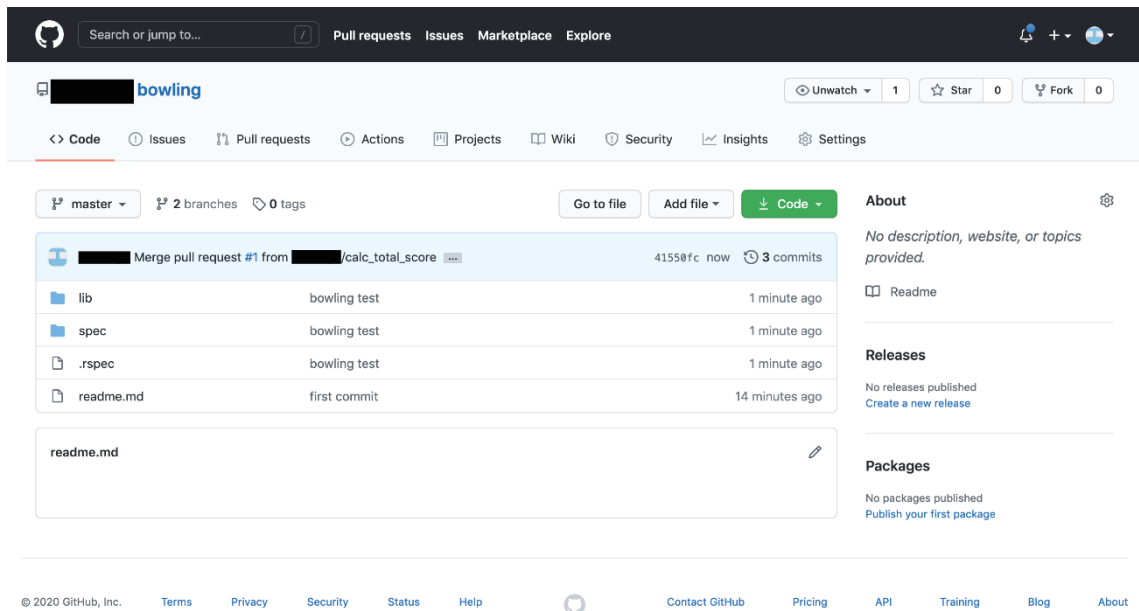
(Wikipedia)



【実習支援ツール】

e-Learning で使用する参考資料やプログラムソースコード管理を行う。

1) GitHub



(GitHub の簡単な説明)

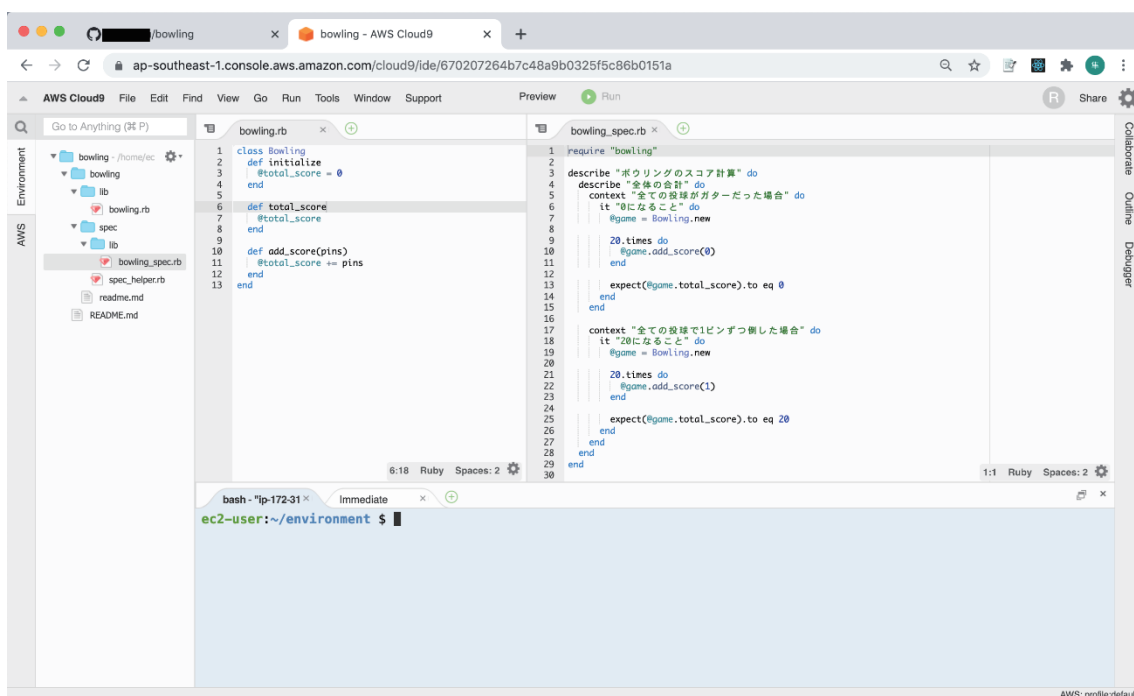
ソフトウェア開発のプラットフォームであり、ソースコードをホスティングする。ソースコードをホスティングすることで複数人のソフトウェア開発者と協働してコードをレ

ビューしたり、プロジェクトを管理しつつ開発を行うことができる。コードのバージョン管理システムには Git を使用する。(Wikipedia)

【プログラム開発環境】

e-Learning で使用するプログラムをコーディング、実行する環境を設定する。

1) AWS、Cloud9



(AWS の簡単な説明)

Amazon Web Services (略称 : AWS) とは、Amazon.com により提供されているクラウドコンピューティングサービスである。ウェブサービスと称しているが、ウェブサービスに限らない多種多様なインフラストラクチャーサービスを提供している。AWS は需要に応じた計算能力を、設定変更のみで速やかに提供出来ることが強みである。クラウドの分野での AWS の世界的シェアは 33% 前後で世界 1 位である。(Wikipedia)

(Cloud9 の簡単な説明)

オンライン IDE (統合開発環境) であり、C、C++、PHP、Ruby、Perl、Python、Node.js を伴う JavaScript、Go などの複数のプログラミング言語をサポートしている。2016 年 7 月に Amazon に買収され、Amazon Web Services (AWS) の一部になった。オンライン IDE として、複数のカーソルを提供することで複数のユーザーからの同時編集を可能にし、

プライベートプロジェクトとパブリックプロジェクトの作成をサポートされた。ユーザーは、ファイルをプロジェクトにドラッグアンドドロップし、タブを使用して複数のファイルを管理することもできるようになった。プロジェクトは、GitHub や Bitbucket などのコラボレーションプラットフォームであるばかりか、さらに Mercurial および Git リポジトリとの統合もまた可能になった。(Wikipedia)

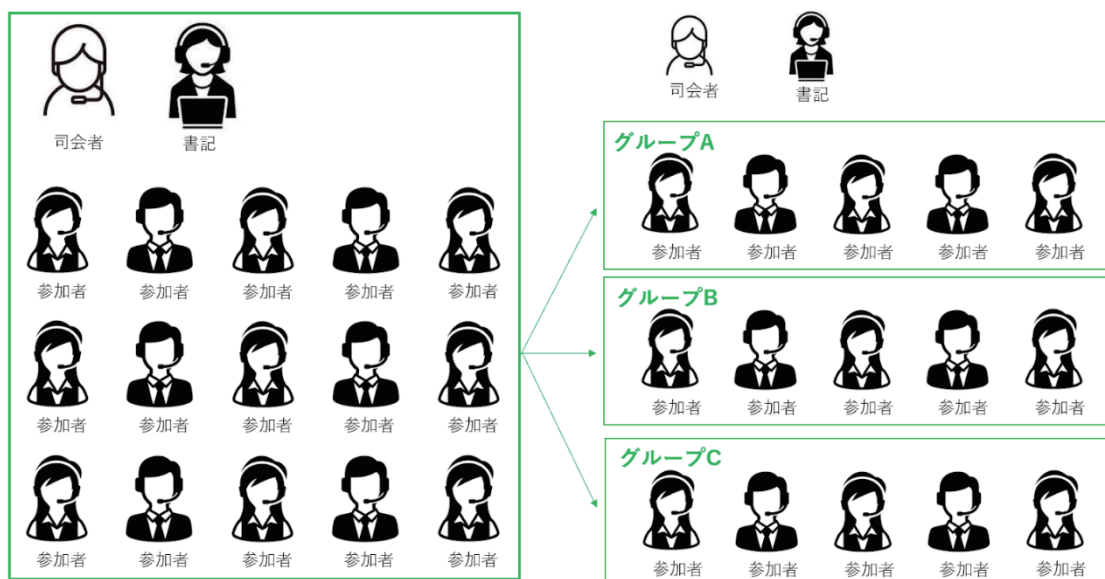
オンラインでのスクーリングで e ラーニングの実習環境の構築等を指導する場合、講師は受講者へビデオ会議アプリのチャット機能を使って各種ツールの URL を提供しながら、各自に登録・設定作業を行わせる。

一般的に、講師はプログラム開発環境を構築するための「受講環境作成手順書」(巻末に例示)をビデオ会議アプリのチャット機能を使って提供し、各自が登録・設定作業をフォローするが、各受講者への個別対応は、ビデオ会議アプリのチームのグループワーク機能を使い、ワンツーマンの対応を行う。

3. オンラインスクーリングでのグループ学習や個別フォロー

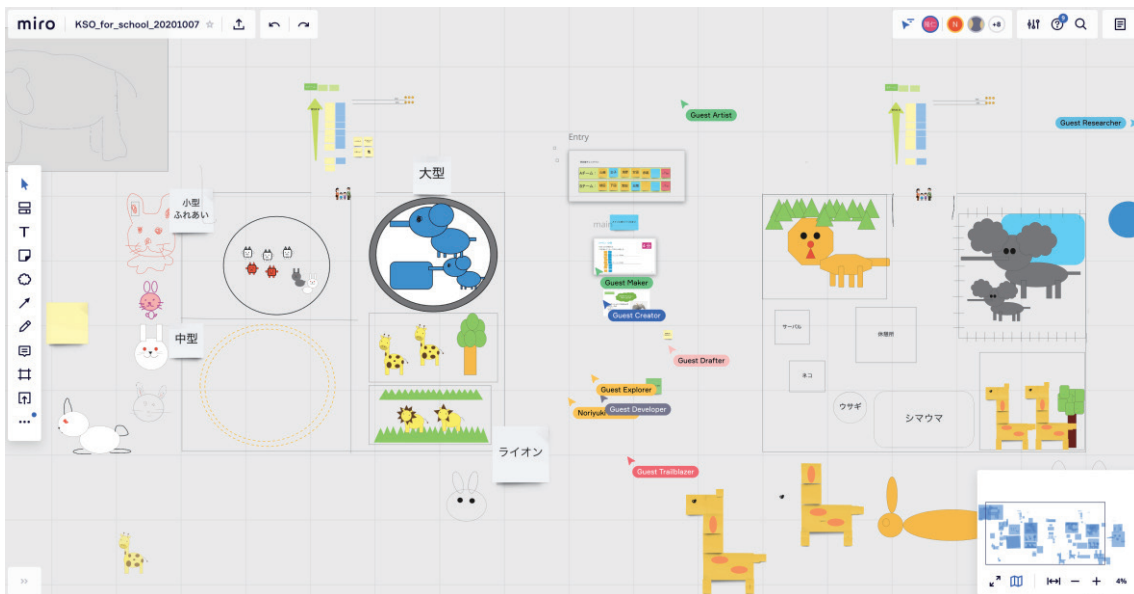
1) Zoom ブレイクアウトルーム機能

セミナー中に、グループワークや個別指導を行いたい場合、ミーティング中にチームごとにブレイクアウトルームを作り、参加者を自動または手動で各グループに割り当てることができる。ブレイクアウトルームの参加者は、オーディオ・ビデオ・画面共有機能を利用し、話し合うことができる。



2) オンラインホワイトボードツールの「MIRO」

オンラインで複数人と共同作業ができるホワイトボードツールである。ブレインストーミングや情報整理など、ミーティング時にホワイトボードを使うようなシーンで利用できる。現実世界のホワイトボードとは異なり、領域に制限がない。マインドマップやタスクボードなど、様々なテンプレートも用意されている。



【MIRO を使ったの実習例】



ふりかえりのための YWT フレームワークを用意しておき、参加者は付箋でそれぞれの感想や意見、アイデアなどをそれぞれの領域に貼りつける。

3) グループワークのプラクティス

アジャイル開発では継続的に価値を生みだし続ける必要があり、それを実現するためには、チーム内での知識共有や共通理解は欠かせない。モブプログラミングはこれらを推進するプラクティスである。**モブプログラミングを実施することによって、製品の品質が高まったり作業効率が向上するなどの効果が得られる。**

(モブプログラミングのやり方)

1 台の PC を複数人で使用しながらプログラミングする。コードを書くドライバー役 1 名とドライバーに指示を出す複数名のナビゲーターとで役割を分担する。ドライバーとナビゲーターは定期的に交代する。

(モブプログラミングのメリット)

- ・ 作業ミスを軽減できる
- ・ コードの質が向上する
- ・ コードの共有がすすむ
- ・ 知識や知見を共有できる
- ・ 作業が効率化される
- ・ 学習スピードが向上する
- ・ コミュニケーションが促進される

(モブプログラミングのデメリット)

- ・ デメリット・大勢でプログラミングするため発言回数が減り、集中力が切れる
- ・ 誰かが気づいてくれるだろう、と他のメンバーに依存しがちになる
- ・ ドライバーとナビゲーター間で経験差が大きい場合、複数のナビゲーターが 1 人のドライバーを攻撃してしまうような構図になることがある

(チーム開発を進める際に講師が行うこと)

開発が全く進められなくなってしまうような状態（ツールが正常に動作しない、など）を除き、開発中に発生するあらゆる問題や課題は可能な限りチーム内で解決してもらうようにすること。講師は、チームがモブプログラミングのメリットを活かしながら「チームとして成果を出せる状態」を作り出せるように、チームを観察する。講師は、スプリントのふりかえりをファシリテーションし、観察によってスプリント中に発見した事実をチームに投げかけて改善を促すなど、スクラムマスターとしての役割の一部を担

う。例えば、チームが小さなタスクを完了したときに、講師が率先して「やったね！」とチームの成功体験を表現することで、その後のチーム内コミュニケーションが円滑になりやすい。

4) Visual Studio Code

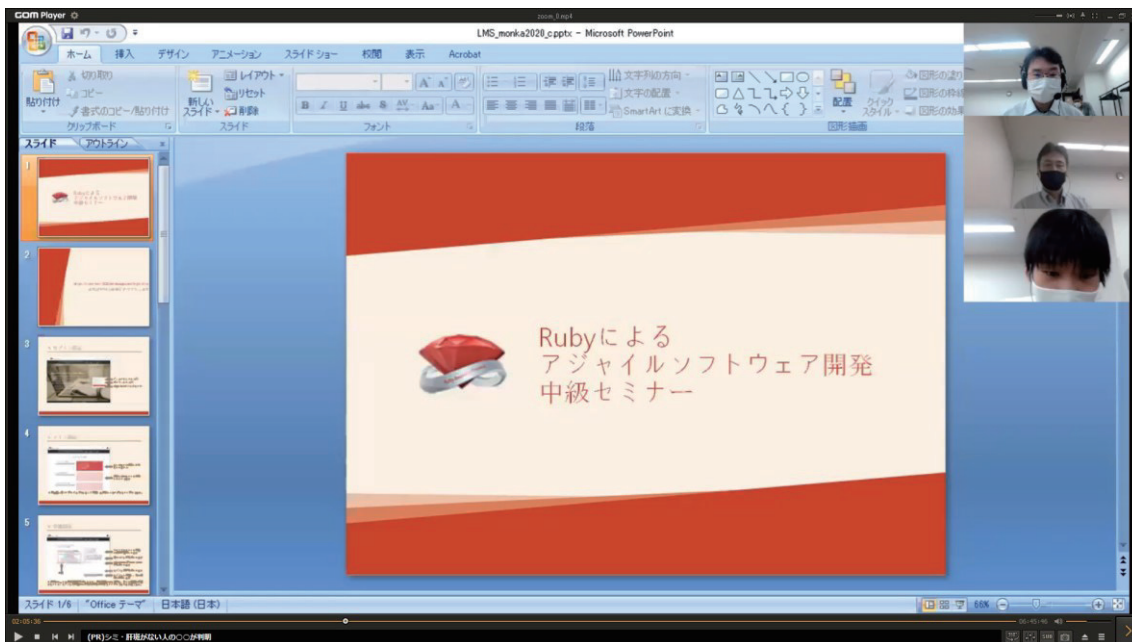
(Visual Studio Code の簡単な説明)

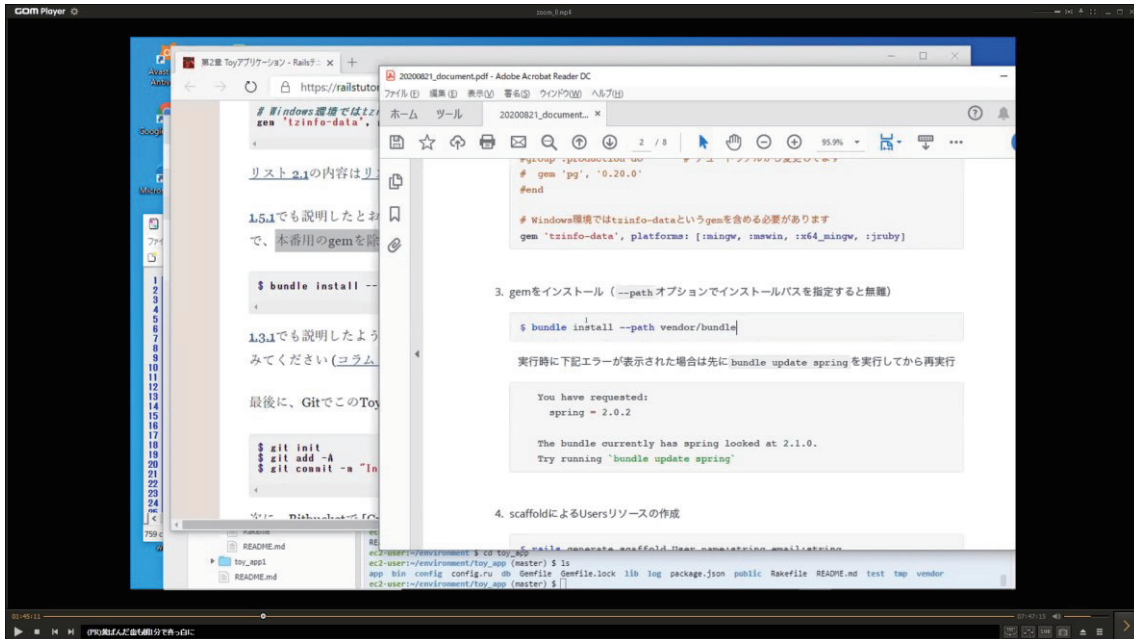
Microsoft が開発した各種 OS に対応するソースコードエディタである。各種プログラミング言語に対応しており、シンタックスハイライトやコード補完、コードリファクタリングなどの機能もあることから利用も多い。拡張機能である Live Share を利用することで、オンラインでのソースコードの共同編集も可能となり、開発者が同じ部屋にいないリモートワーク環境下でもペアプログラミングやモブプログラミングを行うことができる。

4. オンラインスクーリングでの講義

講師はビデオ会議アプリを使って実施する。

ビデオ会議アプリのチャット機能を使って講義中も質問対応が可能となり、受講者の反応をリアルに認知することができる。





5. 演習

講師は受講者へビデオ会議アプリのチャット機能を使って演習問題を配布することができます。また、ビデオ会議アプリのチームのグループワーク機能を使い、チームで課題検討、答合せを実施することができます。

問題1

Ruby における偽の値として正しいものすべてを選択してください。(二つ選択)

- 1. 0
- 2. false
- 3. ""
- 4. nil
- 5. NULL

回答

問題2

以下のコードを実行したときの出力として、正しいものをひとつ選んでください。

```
t850 = -4.0
x = t850 <= -6.0 ? 'snow' : 'rain'
puts x
```

- 1. -4.0
- 2. -6.0
- 3. -10.0
- 4. snow
- 5. rain

回答

問題3

以下のような出力になるコードがあります。
□に入る適切な記述をすべて選択してください。(二つ選択)

```
a = %w(a b c d e f)
p □
# 出力 ["c", "d", "e"]
```

- 1. a[2, 3]
- 2. a[2, 4]
- 3. a[2...-1]
- 4. a[2..-1]
- 5. a[-4, 4]

回答

問題4

以下のコードを実行した時の結果として正しいものをひとつ選択してください。

```
a = 10
b = 10
if !(a - b) then
  puts "hello"
else
  puts "good-bye"
end
```

- 1. hello
- 2. good-bye
- 3. nil
- 4. 何も表示されない

回答

6. 評価

スクーリングに対するアンケートを、チャットを使って受講者に配布し回収することができる。

Rubyオリエンテーションアンケート

次の項目の中で良くなかった／わかりづかったところにチェックを入れて、
質問やコメントをご記入ください。

内容	チェック	質問/コメント
出席者自己紹介	<input type="checkbox"/>	
受講環境の設定	<input type="checkbox"/>	
Slack 設定	<input type="checkbox"/>	
AWS Cloud9 設定	<input type="checkbox"/>	
Github 設定	<input type="checkbox"/>	
eラーニング動画視聴	<input type="checkbox"/>	
Ruby講義	<input type="checkbox"/>	
Ruby演習	<input checked="" type="checkbox"/>	間違いやすいものに補足解説がある方がいいと思いました。
Zoom開催：画質	<input type="checkbox"/>	
Zoom開催：音質	<input type="checkbox"/>	
Zoom開催：サポート (ブレイクアウトルーム)	<input type="checkbox"/>	

その他お気づきの点などございましたらご記入ください。

クレジットカード登録時AWS側でクレジットカードの確認として1円請求されるようなんですが
(調べた感じだと実際はすぐに請求取り消されるようですが)、その説明もあれば安心だと思いまし

ご協力ありがとうございました。

<参考資料>「受講環境作成手順書」

e-learning 事前準備手順

ここでは今回のe-learning講座を進行する上での事前準備を行います。
以下に1.から5.までの手順を示しているため、順に進行をお願いします。

1.Slackでワークスペースへの参加

下記は今回のe-learning用のSlackのURLです。e-learning講座における情報共有の為に活用します。
リンク先から参加をお願いします。参加にはメールアドレスを使用します。
Slackのアカウント登録をされていない方もリンク先から登録が可能です。
https://join.slack.com/t/oic-e-learning2020/shared_invite/zt-g9f4796o-hgrKSp6hPqlmo0i9CYmj3Q

2.GitHubのアカウント作成

e-learningのコンテンツ内で使用するアカウントを作成します。
アカウントを既に取得されている方は、そちらを使っても構いません。
※プランは\$0のフリープランを選択してください
作成方法は別紙「GitHubアカウント作成」をご参考ください

3.AWSアカウント登録

e-learningを進行するうえで必須となるアカウントです。
AWSのアカウント登録には、以下のものが必須となりますのでご準備した上で進めてください。
・クレジットカードまたはデビットカード
・SMSまたは携帯電話
使用方法によっては利用料金が発生する可能性があります。
登録前に必ず²ex.補足情報をご覧ください。
下記のリンクにAWSのアカウント取得手順が記載されていますので、そちらをご参考に進めてください。
AWSアカウント取得手順説明：<https://aws.amazon.com/jp/register-flow/>

4.Cloud9環境構築

e-learningを進行する為に使用する開発環境です。
設定内容によっては料金が発生するものがありますので、必ず手順通りに行ってください。
手順は別紙「Cloud9環境構築」をご参考ください

5.Ruby on Rails 環境構築

3のCloud9環境構築の終了直後からの続きになります。
Ruby on Railsの環境構築を行います。
手順は別紙「Ruby on Rails 環境構築」をご参考ください

本資料について

- 本書はRuby言語プログラムの基礎知識についての解説資料です。
- 他言語の使用経験者を対象としています。よって、各項目はRuby独特の機能及び、Rubyの構文のみの記載としており、「ループとは何か?」という様なプログラミングの基本的概念についての解説は本書では行いません。
- 各学習項目には対応するリファレンスのページを記載しています。本書記載内容以上の詳細については、受講者各自にてリファレンスの参照をお願い致します。

第1回目受講内容

- Rubyプログラム基本事項
- 変数・定数
- 演算子・条件式・修飾子
- ループ
- 文字列
- 配列・ハッシュ
- イテレータ
- 例外処理
- オブジェクト
- メソッド・レシーバ
- 演習と解説

Rubyプログラム基本事項

・拡張子, 基本構文

・helloworld.rb

・ファイル拡張子: rb

```
puts "Hello World!"
```

・行末記号: セミコロン(;)は省略可。

```
#puts "Good Morning!"
```

・コメントその1: 1行コメント

```
=begin
```

#以降がコメントと判断される。

```
puts "Good"
puts "Bye!"
```

・コメントその2: 複数行コメント

```
=end
```

=beginと=endで囲んだ

範囲はコメントと判断される。

変数・定数①

```
1
-12
123.45
1.2e-3
0x1f
0o37
```

・数値リテラル

```
"Good Morning¥n"
'Good Evening¥n'
```

・文字列リテラル:

“”は拡張表記・式展開(後述)が有効

‘’は拡張表記・式展開(後述)が無効

```
/^Ruby$/
```

・正規表現リテラル

/で囲んだ文字列は正規表現

変数・定数②

・変数, 定数の表記例

<code>localnum = 1</code>	←	・ローカル変数
<code>@inststr = "ruby"</code>	←	・インスタンス変数
<code>@@ clasnum = 100</code>	←	・クラス変数
<code>\$globalstr = "GLBL"</code>	←	・グローバル変数
<code>FIXEDNUM = 777</code>	←	・定数

※Rubyでは変数の『型』の宣言は不要。プログラム実行時、Rubyインタプリタが各リテラルを自動的に解釈し、型を決定する。

変数・定数③

・Rubyにおける真偽判断

・擬似変数false と nil

Rubyでは条件式などの判定の際、擬似変数false, nilのみを『偽』と判断し、それ以外のすべては『真』と判断する。
(0も『真』と判断されることに注意)

※擬似変数・・・Rubyが提供する特殊変数。定数の様な働きをする。

※nil・・・他言語で言うところのNULLに当たる。

尚、RubyにNULLは存在しない。

演算子①

・算術演算子

+	・・・加算	$a = b + c$
-	・・・減算	$a = b - c$
*	・・・乗算	$a = b * c$
**	・・・べき乗	$a = b ** c$
/	・・・除算	$a = b / c$
%	・・・剰余	$a = b \% c$
=	・・・代入	$a = b$

演算子②

・代入演算子

+=	・・・加算後代入	$a += b$	($a = a + b$ と同等)
-=	・・・減算後代入	$a -= b$	($a = a - b$ と同等)
*=	・・・乗算後代入	$a *= b$	($a = a * b$ と同等)
/=	・・・除算後代入	$a /= b$	($a = a / b$ と同等)
%	・・・剰余を代入	$a \% = b$	($a = a \% b$ と同等)

※ちなみに、rubyでは

インクリメント($a++$, $++a$),デクリメント($a--$, $--a$)共に存在しないのでプログラミング時は注意。

演算子③

・比較演算子

==	・・・等しい	$a == b$
<	・・・小なり	$a < b$
>	・・・大なり	$a > b$
<=	・・・以下	$a <= b$
>=	・・・以上	$a >= b$
!=	・・・等しくない	$a != b$

演算子④

・論理演算子

&, &&	・・・and	$(a >= 1) \&\& (b <= 0)$
,	・・・or	$(a >= 1) \ \ (b <= 0)$
!	・・・not	$!(a == 1)$

※上記とは別にand, or, not演算子も使用可能。

$(a >= 1) \text{ and } (b <= 0)$

$(a >= 1) \text{ or } (b <= 0)$

$\text{not}(a == 1)$

演算子⑤

・範囲演算子

1..4 1から4まで(1, 2, 3, 4)

1...4 1から3まで(1, 2, 3)

“a”..”d” aからdまで(a, b, c, d)

“a”...”d” aからcまで(a, b, c)

※Rubyでは範囲を指定したい際、演算子『..』と『...』が使用可能。順序が昇順に特定可能な整数やアルファベットに対して範囲を指定できる。『..』は終端要素を含むが、『...』は終端要素を含まない。

条件式・修飾子①

・Rubyにおける制御文

Ruby言語プログラムにおける各種制御文について解説する。

①if文の構文

<pre>if (条件式) then 処理</pre>	<p>← 条件式には他言語同様、判定結果が真または偽となる式を記述する。(例: $n < 5$, $m == true$ 等)</p> <p>またthenの記述は省略可能。</p>
<pre>elsif (条件式) then 処理</pre>	<p>← 条件の後に実行する処理の部分を『ブロック』と言う。</p>
<pre>elsif (条件式) then 処理</pre>	<p>← ifの条件、elsifの条件いずれにも該当しない場合、elseの処理内容が実行される。</p>
<pre>else 処理</pre>	<p>← ifの条件、elsifの条件いずれにも該当しない場合、elseの処理内容が実行される。</p>
<pre>end</pre>	<p>← if文の最後は”end”で閉じる。</p>

条件式・修飾子②

②unless文の構文

```
unless (条件式) then
  条件式の結果が『偽』の時
  実行される処理
else
  条件式の結果が『真』の時
  実行される処理
end
```

・条件式には判定結果が真または偽となる式を記述する。(例: $n < 5$, $m == \text{true}$ 等)
またthenの記述は省略可能。

・unless文は実行されるブロックがif文とは逆に
なっている制御文。
また、if文におけるelsifはunlessには存在しない。

条件式・修飾子③

③if修飾子とunless修飾子

if, unlessは条件を処理の後に記述する事も可能。

・if修飾子の構文

```
処理 if (条件式)
```

または

```
begin
  処理
end if (条件式)
```

・条件式の判定結果が『真』の場合、処理が実行される。

・unless修飾子の構文

```
処理 unless (条件式)
```

または

```
begin
  処理
end unless (条件式)
```

・条件式の判定結果が『偽』の場合、処理が実行される。

条件式・修飾子④

④case文の構文

```
case (式)
```

```
when 値1 then
```

```
  処理
```

```
when 値2 then
```

```
  処理
```

```
when 値3 then
```

```
  処理
```

```
else
```

```
  処理
```

```
end
```

・値1, 値2, 値3には数値と文字列が指定可能。
また、thenの記述は省略可能。

・式と値が一致した処理ブロックが実行される。

・いずれのwhenの条件にも一致しない場合、
elseの処理のブロックが実行される。

ループ①

①for文の構文

```
for (変数) in (配列, 範囲等) do
```

```
  処理
```

```
end
```

・Rubyにおけるfor文では、ループの1順毎に
(配列, 範囲等)に指定したオブジェクトから要素を
1つずつ取り出し(変数)に代入した後、処理のブロック
が実行される。
なお、doの記述は省略可能

・記述例: 1から5までの合計を求めるプログラム

```
n = 0
```

```
for i in 1..5 do
```

```
  n += i
```

```
end
```

・ループの1順毎に範囲オブジェクト1..5から
要素が順に取得され、変数iに代入されていく。

ループ②

②while文の構文

```
while (条件式) do
```

```
    処理
```

```
end
```

・条件式には他言語同様、判定結果が真または偽となる式を記述する。(例: $n < 5$ 等)
またdoの記述は省略可能。

ループ③

③until文の構文

```
until (条件式) do
```

```
    処理
```

```
end
```

・条件式には他言語同様、判定結果が真または偽となる式を記述する。(例: $n < 5$ 等)
またdoの記述は省略可能。

ループ④

④while修飾子とuntil修飾子

while, untilは条件を処理の後に記述する事も可能。

・ while修飾子の構文

```
begin
  処理
end while (条件式)
```

・まず、begin～endで囲んだブロックの処理が実行され、その後条件式の判定が行われる事に注意。

・ until修飾子の構文

```
begin
  処理
end until (条件式)
```

・まず、begin～endで囲んだブロックの処理が実行され、その後条件式の判定が行われる事に注意。

ループ⑤

⑤ループ処理の制御文

for文, while文等のループ処理の特殊な制御を実行する命令を下記する。

・break: ループ処理を中断

```
for (条件式)
  while (条件式)
    処理
    break ←
    処理
  end
  n += 1
end
puts n
```

・breakが実行されると、1番近いブロックの終わりへジャンプする。
(このプログラムの場合ならば、n += 1の行へジャンプ。)

ループ⑥

- `next`: 以降の処理を飛ばして次のループ処理を実行

```
for i in 2..5
  処理A
  next ←
  処理B
end
```

- `next`が実行されると、現在のループ処理を中断し、次のループ条件でループ処理が実行される
(例: `i` の値が2の時に`next`が実行された場合、処理Bの実行は行われず、`i = 3`のループ処理が実行される。)

- `redo`: 以降の処理を飛ばしてもう一度同じ条件でループ処理を実行

```
for i in 2..5
  処理A
  redo ←
  処理B
end
```

- `redo`が実行されると、現在のループ処理条件でもう一度ループ処理が実行される
(例: `i` の値が2の時に`redo`が実行された場合、処理Bの実行は行われず、もう一度 `i = 2`の条件でループ処理が実行される。)

文字列

文字列データの場合(Stringクラス)

```
str1 = "ruby" ←
```

- 文字列データ`str1`を定義。
`str1`はStringクラスのオブジェクトとなる。

```
puts str1.length # バイト数を表示 ←
```

- Stringクラスのメソッド`length()`を実行。
`length()`は文字列の長さ(バイト数)を返却するメソッド。

```
# 先頭を大文字、以降を小文字に変換
puts str1.capitalize ←
```

- Stringクラスのメソッド`capitalize()`を実行。
`capitalize()`は先頭文字を大文字、先頭以外を小文字に変換するメソッド。

```
# オブジェクトIDを表示
puts str1.object_id ←
```

- Stringクラスのメソッド`object_id()`を実行。
オブジェクト`str1`のオブジェクトIDが表示される。

配列・ハッシュ①

・配列の表記

```
list = ["one", 2, "three"]
```

- ・配列の定義。[]の中に要素を記述する。
Rubyでは型の違う要素の混在が可能。

```
puts list[0]
```

- ・任意の要素の参照

```
list[1] = "two"
```

- ・任意のindexへの代入。

配列・ハッシュ②

・配列の表記その2

```
list = [1, "two", 3]
```

```
puts list [-1]
```

- ・Rubyでは、末尾よりindexを-1, -2, ...と指定する事が可能。

(この配列の場合、list[-1] ⇒ 3, list[-2] ⇒ "two", list[-3] ⇒ 1となる。)

```
list [5] = "six"
```

- ・配列のサイズを超えたindexの指定も可能。
この行の実行後、配列listは以下ようになる。

```
list = [1, "two", 3, nil, nil, "six"]
```

※ 足りない要素は自動的にnilで埋められる。

配列・ハッシュ③

プログラム中で配列を定義した場合を考える。

```
list1 = [1, 2, 3, 4, 5]
list2 = ["a", "b", "c"]
list3 = ["a", "b", "c"]

puts list1.size # 要素数を表示

list1.clear # 全要素削除

puts list2.object_id #IDを表示
puts list3.object_id #IDを表示
```

- ・配列データlist1, list2, list3を定義。
この際、自動的に(明示的に型宣言をしなくても) list1, list2, list3はArrayクラスのオブジェクトとなる。
- ・Arrayクラスのメソッドsize()を実行。
size()は自己の要素数を返却するメソッド。
(オブジェクトの情報)
- ・Arrayクラスのメソッドclear()を実行。
clear()は自己の要素を全て削除するメソッド。
(オブジェクトの機能)
- ・Arrayクラスのメソッドobject_id()を実行。
object_id()はオブジェクトのユニーク値である
オブジェクトIDを返却するメソッド。list2, list3は内容が
同じでありながら、それぞれ違うIDを表示する。
(オブジェクトの一意性)

配列・ハッシュ④

・ハッシュの表記

```
hash = {"name" => "ruby", "ver" => 1.8,
        255 => "ff"}

hash["ver"] = 1.9

n = hash[255]
```

- ・ハッシュの定義。{ }の中に
キー => 値 の形で要素を記述する。
- ・値の代入。
- ・値の参照。この場合変数nの
値は文字列“ff”になる。

配列・ハッシュ⑤

・ハッシュの表記その2

```
hash = {"name" => "ruby", "ver" => 1.8,
        255 => "ff"}
```

```
hash["num"] = 12345
```

```
Hash = {apple: 100, banana: 200}
```

・ハッシュの追加。

ハッシュhashの状態は以下のようになる。

```
hash={"name" => "ruby", "ver" => 1.8, 255 => "ff",
      "num" => 12345}
```

・シンボル形式が一般的

配列・ハッシュ⑥

プログラム中でハッシュを定義した場合を考える。

```
hash1 = {"key1" => 100,
         "key2" => "ruby",
         "key3" => "Rails"}
```

```
puts hash1.length # 要素数を取得
```

```
# 指定したキーを持つかチェック
```

```
puts hash1.has_key?("key2")
```

```
# 指定した値を持つかチェック
```

```
puts hash1.has_value?(100)
```

・ハッシュデータhash1を定義。

hash1はHashクラスのオブジェクトとなる。

・Hashクラスのメソッドlength()を実行。

length()はハッシュの要素数を返却するメソッド。

・Hashクラスのメソッドhas_key?()を実行。

has_key?()は指定したキーを持つか否かを
チェックするメソッド。(true / falseで返却)

・Hashクラスのメソッドhas_value?()を実行。

has_value?()は指定した値を持つか否かを
チェックするメソッド。(true / falseで返却)

イテレータ①

・イテレータとは

Rubyには繰り返し処理の実装に特化した『イテレータ』というメソッドが標準で提供されている(*1)。

特定の処理を繰り返し実行するプログラムの実装が必要な場合、イテレータを有効に活用する事で、for文やwhile文などの制御文を用いるよりも少ない記述量で、繰り返し処理の実装を行う事が可能となる(*2)。

*1・・・イテレータ(iterator: 反復子)。ブロック付きメソッドとも呼ばれる。各種イテレータは、Numeric, Array等、各種標準クラスのインスタンスメソッドとして実装されている。

*2・・・記述量の削減の他、for文を用いた際などに生じがちな「条件文の定義ミス」による不具合も回避できる。このような特徴から、イテレータはRubyプログラムにおいて得に有用性が高い。

イテレータ②

・代表的なイテレータ

以下にRubyが提供しているイテレータの一部を例示すると共に、イテレータの基礎についての解説を行う。

①timesメソッド

・timesメソッドの使用例

```
3.times {
  puts "Ruby Seminar"
  puts "Let's Study !!"
}
```

※実行結果

```
Ruby Seminar
Let's Study !!
Ruby Seminar
Let's Study !!
Ruby Seminar
Let's Study !!
```

・イテレータtimesの使用例。

本プログラムは{ }内の処理(2つのputs命令)を、timesメソッドのレシーバとして設定した数値(3)の回数だけ繰り返し実行する。

timesメソッドを含む各種イテレータの構文は、このように

オブジェクト名.メソッド名 { 繰り返したい処理(=ブロック) }

となる。(尚、{ } は do end でも代用可。)

timesメソッドは整数値型クラスIntegerのインスタンスメソッドとして実装されている。この様にRubyの各種基本クラスには、各々のクラスの特徴に適したイテレータが実装されている。

イテレータ③

②eachメソッド

・eachメソッドの使用例

```
list = [ 1, 2, 3, 4, 5]
list.each { |i|
  puts "#{ i ** 2 }"
}
```

- ・ 配列型クラスArrayのeachメソッドの実行。
本プログラムは配列listの各要素の2乗を順に出力する。

※実行結果

```
1
4
9
16
25
```

{ }内の最初に | | に囲まれて記載されている変数 i は『**ブロック変数**』
と言い、レシーバに設定したオブジェクトの値を順に受け取る。(この
プログラムの場合、繰り返し処理の1順毎に配列 list の要素が先頭から
順にセットされる。)
ブロック変数に設定される値はイテレータ毎の仕様により異なるが、
基本的に、「繰り返し処理の1順毎に異なる値が設定される」変数である。
(前頁のtimesメソッドはブロック変数を必要としないイテレータである為、
記載を省略した。このように、ブロック変数は省略されることがある。)

ブロック引数がある場合のイテレータの構文は以下。

```
オブジェクト名.メソッド名 { | ブロック変数1, ブロック変数2... |
  ブロック }
```

イテレータ④

③その他のイテレータ

前述したtimes, eachの他、Rubyの各基本クラスには様々なイテレータが実装されている。
各種基本クラスについての仕様を確認し、実装されているイテレータの仕様を十分に理解する事で、
Rubyならではの手法で生産効率を向上させることが可能である。

例外処理①

・例外と例外処理

プログラム中で例外が発生するとプログラムが終了してしまう。

例外処理を記述することにより、プログラムの継続が可能となる。

・例外処理構文

```
begin
  例外が起こる可能性がある処理
rescue
  例外発生時の処理
else
  例外が発生しなかった場合の処理
ensure
  例外の有無に関わらず実行する処理
end
```

・ファイルオープン処理等、例外が発生する可能性がある処理をbegin以降に記述。

・begin以降のブロックで例外が発生した際に実行されるブロック。

・begin以降のブロックで例外が発生しなかった際に実行されるブロック。なお、例外非発生時の処理が必要無い場合はelse及びelse以降のブロックの記述は省略が可能。

・rescueまたはelseのブロックの後に実行されるブロック。必要無い場合、ensure及びensure以降のブロックの記述は省略が可能。

例外処理②

・例外処理記述例(ファイルオープンエラーの場合)

```
begin
  puts "open sample.txt"
  File.open("sample.txt")
rescue
  puts "ERROR: #!"
else
  puts "ファイルを開きました。"
ensure
  puts "プログラムを終了します。"
end
```

・ファイルsample.txtが存在しない状態で、ファイルオープンを試みる。

・エラー発生時の処理。エラー内容を表示する。(エラー発生時、特殊変数\$!にはエラー内容が代入される。)

実行結果

```
open sample.txt"
ERROR: No such file or directory – sample.txt
プログラムを終了します。
```

オブジェクト①

・Rubyにおけるオブジェクトの概念

Ruby言語プログラムの最大の特徴として、
『Rubyプログラム中の(ほぼ)全てのデータはオブジェクトである』
という点が上げられる。

『全てのデータはオブジェクトである』とは、プログラム中で使用されている各種データ(=変数, 配列, 正規表現, クラス, 演算子等)は、それぞれ単なる値では無く、『何らかのクラスのオブジェクトである。』
という意味である。

オブジェクト②

・『全てのデータはオブジェクトである』とは 具体的にはどういう事か？

「オブジェクト」とはそもそも、「情報と機能の塊」である。
つまり、Rubyプログラム中でデータを定義(例: num = 10)した際、
自動的にそのデータ(変数num)は

- ・自己の内部状態を持つ。(オブジェクトのデータ)
- ・自己を操作するためのメソッドを持つ。(オブジェクトの機能)
- ・自己と他を区別する。(オブジェクトの一意性)

という特性を持つようになる。

メソッド・レシーバ①

・メソッドの定義

・メソッド定義の構文

```
def methodName(param1, param2)
  x = param1 + param2
  return x
end
```

- ・メソッドの書き始めはdefで始まり、メソッド名(仮引数名1, 仮引数名2, ...)と記述していく。戻り値の型や仮引数の型の記述はRubyでは不要。
- ・return文でメソッドを終了し、戻り値を返却する。return文を省略した場合、最後に記述した値が戻り値となる。
- ・メソッド定義の終わりはendで終わる。

・引数の無いメソッドの場合

```
def method()
  処理
end
```

- ・引数の無いメソッドの場合は()の省略が可能。

メソッド・レシーバ②

・メソッドの呼び出し

・引数の有るメソッドの場合

```
def methodName(param1, param2)
  x = param1 + param2
  return x
end
```

```
n = methodName(2, 3)
```

- ・メソッド名(実引数)で呼び出す。

・引数の無いメソッドの場合

```
def method()
  処理
end
```

```
n = method()
n = method
```

- ・引数の無いメソッドの場合は()の省略が可能。

メソッド・レシーバ③

・特殊な引数を持つメソッドの定義

・デフォルト引数

```
def method(param1, param2 = 4)
```

・仮引数の後に = をつけて仮引数のデフォルト値を指定。

```
  x = param1 + param2
```

```
  return x
```

```
end
```

```
n = method(5, 3)
```

・メソッドの呼び出し時、第1, 第2引数ともに指定した場合、デフォルト引数の値は用いられない。

(この場合、methodでは 5 + 3 が実行される)

```
m = method(5)
```

・メソッドの呼び出し時、引数を省略するとデフォルト値が適用される。

(この場合、methodでは 5 + 4 が実行される)

メソッド・レシーバ④

・可変長引数

```
def method(param1, *param2)
```

・最後の仮引数の頭に * をつける事でメソッドの引数の数を可変にすることが出来る。

(※尚、* をつけることが可能なのは最後の仮引数のみ)

```
  x = param1
```

```
  for i in param2
```

```
    x += i
```

```
  end
```

```
  return x
```

```
end
```

```
n = method(5, 4)
```

・メソッドの呼び出し側で、可変長引数以降に指定された値は * をつけた変数に配列として格納される。

このプログラムの場合、

method(5, 3)の場合はparam2の値は配列[4]に、

method(5, 4, 3)の場合はparam2の値は配列[4, 3]に、

method(5, 4, 3, 2, 1)の場合はparam2の値は配列[4, 3, 2, 1]

```
m = method(5, 4, 3)
```

```
l = method(5, 4, 3, 2, 1)
```

メソッド・レシーバ⑤

・ブロック引数

```
def method(param1, &param2)
  i = 0
  while i < param1
    param2.call()
    i += 1
  end
end
```

・仮引数の頭に & をつける事でブロック引数となる。
ブロック引数には実引数としてブロック(=処理のかたまり)を渡す。

・メソッド内でブロック引数名.call()と実行する事で引数として受け取ったブロックの内容が実行される。

```
method(3){ puts "Ruby Seminar" }
```

・{ }またはdo~endで、引数として渡したいブロックを定義する。

実行結果

```
Ruby Seminar
Ruby Seminar
Ruby Seminar
```

第2回目受講内容

- クラス・インスタンス
- アクセスメソッド
- 継承・カプセル化
- モジュール・Mix-in
- シンボル
- 予約語
- 正規表現
- ライブラリ、Timeクラス
- コマンドライン引数
- ファイル入出力、ディレクトリ操作
- 演習と解説

02/51

クラス①

・クラスの定義

他のオブジェクト指向言語同様、Rubyのクラスの定義が可能であり、クラスそのものの概念も他言語と同等である。

・クラス定義の構文

```
class クラス名
  :
  変数、メソッドの定義等
  :
end
```

・class クラス名 ~ endの形式でクラスの定義が可能。
命名規則として、クラス名の先頭は必ず大文字にする事。

クラス②

・クラス中の変数・メソッド定義

クラス内には、変数・定数・メソッドの定義が可能。

以下に各々の構文と解説を記載する。

①変数, 定数の構文

```
class クラス名
  VERSION = "1.8"
  @insstr
  @@clsnum
end
```

・定数

・インスタンス変数。先頭に@をつけて定義する。
インスタンス変数はクラス中でグローバルに使用
できる変数。

・クラス変数。先頭に@@をつけて定義する。
クラス変数は同一クラスから生成された全ての
オブジェクトで共有される変数。

クラス③

②メソッド定義の構文(インスタンスメソッド)

```
class クラス名
  def method( param )
    :
  end
end
```

・一般的なメソッドの定義。

def メソッド名 ~ end で囲まれた箇所がメソッドとなる。
後述するクラスメソッドと比較し、
一般的なメソッドを『インスタンスメソッド』と呼ぶ。

※インスタンスメソッドの呼び出し

```
class Testcls
  def methodA( param )
    :
  end
end

obj = Testcls.new

obj.methodA( 100 )
```

・インスタンスメソッドの呼び出し方。

オブジェクト名.メソッド名の形式で記述する。

尚、メソッド実行の際、「.」の左に記述するものを

『レシーバ』と呼ぶ。(この場合、変数objがレシーバ。)

クラス④

③メソッド定義の構文(クラスメソッド)

```
class Aclass
  def Aclass.method( param )
    :
  end
end
```

・クラスメソッドの定義。

def クラス名.メソッド名 ~ end で囲まれた箇所がクラスメソッドとなる。オブジェクトに依存しない処理を実装するメソッドは、クラスメソッドとして定義する事。

尚、クラスメソッドの中でインスタンス変数を扱う事は出来ない。

※クラスメソッドの呼び出し

```
class Testcls
  def Testcls.methodA( param )
    :
  end
end
```

Testcls.methodA(100)

・クラスメソッドの呼び出し方。

レシーバとしてクラスを指定する。

クラス⑤

④メソッド定義の構文(initializeメソッド)

```
class Aclass
  def initialize
    :
  end
end
```

・ initializeメソッドの定義。

initializeメソッドはクラスの初期化処理を行うためのメソッド。クラス外よりnewメソッドが実行された際、自動的にinitializeメソッドが実行される。

※ initializeメソッドの呼び出し

```
class Testcls
  @insnum
  def initialize( param )
    @insnum = param
  end
end
```

Testcls.new(10)

・ newを実行する事で、initializeメソッドが実行される。また、newに指定した実引数がinitializeメソッドの仮引数に渡される。

クラス⑥

・変数・定数へのアクセス

クラス外より、クラス内に定義した各種変数・定数へアクセス(参照・代入)する際の手法を解説する。

①クラス外からの定数の参照

```
class Ruby
  VERSION = "1.8"
  :
end
```

```
puts Ruby::VERSION
```

← 定数は外部より直接参照が可能(インスタスの生成は不要)。参照の際はクラス名::定数名の形式で構文を記述する。

クラス⑦

②クラス外からの、クラス変数・インスタンス変数の参照

```
class Ruby
  @insnum
  @@clsnum = 10

  def initialize( param )
    @insnum = param
  end
```

```
  def insnum
    @insnum
  end

  def clsnum
    @@clsnum
  end
```

```
end
obj = Ruby.new(100)
puts obj.insnum
puts obj.clsnum
```

← 他のオブジェクト指向言語同様、
クラス変数、インスタンス変数の値を返却値とする
メソッド(=ゲッター)を定義し、それを利用する事で
各変数の参照を行う。

← インスタンス変数@insnumのゲッターを呼び出し
← クラス変数@@clsnumのゲッターを呼び出し

クラス⑧

②クラス外からの、クラス変数・インスタンス変数の代入

```
class Ruby
  @insnum
  @@clsnum

  def insnum=( param )
    @insnum = param
  end

  def clsnum=( param )
    @@clsnum = param
  end
end

obj = Ruby.new

obj.insnum=(123)
obj.clsnum=(777)
```

・他のオブジェクト指向言語同様、
クラス変数、インスタンス変数の値に仮引数の値を設定するアクセス(=セッター)を定義し、それを利用する事で各変数の参照を行う。
尚、Rubyでは慣例として、セッターのメソッド名を“変数名=”にする。

・インスタンス変数@insnumのセッターを呼び出し
・クラス変数@@clsnumのセッターを呼び出し

アクセスメソッド①

・アクセスメソッドを用いた変数アクセス

Rubyでは、クラス内に定義した各種変数へアクセスする方法として、前述のゲッター、セッターを定義する方法の他、『アクセスメソッド(*1)』を使用する手法も存在する。

アクセスメソッドを用いる事で、ゲッター、セッターの定義を簡略化する事が可能となる。(生産性の向上)

*1・・・資料によってはアクセサとも呼称する。

アクセスメソッド②

・アクセスメソッドの使用例

以下にアクセスメソッドの使用例と概要を記載する。
アクセスメソッドには用途に応じて3種類が存在する。

①アクセスメソッドattr_readerの使用例

```
class Testcls
  attr_reader :name

  def initialize( param )
    @name = param
  end
end

obj = Testcls.new("ruby")

puts obj.name
```

- ・アクセスメソッドは
アクセスメソッド インスタンス変数と同名のシンボル
という構文で用いる。
アクセスメソッドattr_readerは指定したインスタンス変数の参照(読み取り)を可能とする。attr_reader :nameと定義する事で、明示的にインスタンス変数@nameのゲッターを定義しなくても、@nameの値がクラス外から参照可能となる。
- ・インスタンス変数@nameの値を参照。

アクセスメソッド③

②アクセスメソッドattr_writerの使用例

```
class Testcls
  attr_writer :name

  def name
    @name
  end
end

obj = Testcls.new
obj.name = "Ruby on Rails"

puts obj.name
```

- ・アクセスメソッドattr_writerは指定したインスタンス変数の代入(書き込み)を可能とする。attr_writer :nameと定義する事で、明示的にインスタンス変数@nameのセッターを定義しなくても、@nameへクラス外から代入が可能となる。
- ・インスタンス変数@nameへの代入。アクセスメソッドを用いることで、オブジェクト名.インスタンス変数名という形式で値の代入を行う事が出来る。
- ・ゲッターメソッドname()の実行

*実行結果

```
Ruby on Rails
```

アクセスメソッド④

③アクセスメソッドattr_accessorの使用例

```
class Testcls
  attr_accessor :name
end
```

・アクセスメソッドattr_accessorは指定したインスタンス変数の参照と代入(読み・書き)の両方を可能とする。

attr_accessor :nameと定義する事で、

明示的にインスタンス変数@nameのゲッター・セッターを定義しなくても、@nameのクラス外からの参照・代入が可能となる。

```
obj = Testcls.new
```

```
obj.name = "Ruby Seminar"
```

・インスタンス変数@nameへの代入。

```
puts obj.name
```

・インスタンス変数@nameの値を参照。

*実行結果

```
Ruby Seminar
```

アクセスメソッド⑤

・アクセスメソッドまとめ

3種のアクセスメソッドについての概要を以下に纏める。

・attr_reader(読み取りを許可)

```
class Testcls
  attr_reader :name
end
```

両者は等しい

```
class Testcls
  def name
    @name
  end
end
```

・attr_writer(書き込みを許可)

```
class Testcls
  attr_writer :name
end
```

両者は等しい

```
class Testcls
  def name=( param )
    @name = param
  end
end
```

・attr_accessor(読み書き両方を許可)

```
class Testcls
  attr_accessor :name
end
```

両者は等しい

```
class Testcls
  def name
    @name
  end
  def name=( param )
    @name = param
  end
end
```

継承・カプセル化①

・Rubyにおける継承

他のオブジェクト指向言語同様、Rubyでもクラスの継承は可能。
但し、Rubyではクラスの**多重継承は不可**。

・クラス継承の構文

```
class Subclass < Superclass
  :
  :
  :
end
```

・サブクラス名(子) < スーパークラス名(親)
の形式で定義する。

継承・カプセル化②

・オーバーライド

他のオブジェクト言語同様にRubyでもスーパークラスで実装されている
メソッドのオーバーライドが可能。

以下にオーバーライドの記述例を記載する。

・オーバーライドの記述例

```
class Eng
  def hello
    print("Hello !! ¥n")
  end
end

class Jap < Eng
  def hello
    print("こんにちは！¥n")
  end
end
```

・スーパークラスEngのメソッドhelloを
サブクラスJapで再定義した場合の
記述例

継承・カプセル化③

・オブジェクト情報の参照

クラス中にて、特殊な命令を実行する事により、サブクラスおよびスーパークラスの情報を取得する事が可能。

・クラス自身の情報参照の実行例(selfの使用)

```
class Testclass
  def methodA
    puts "hello world !!"
  end

  def methodB
    puts "methodA call"
    self.methodA
  end

  def methodC
    puts self.object_id
  end
end

obj = Testclass.new

puts obj.object_id
obj.methodB
obj.methodC
```

・オブジェクト自身やメソッド自身の情報を参照する際は、『self』を用いる。ここでは自身(= Testclassクラス)のメソッドmethodAを呼び出している。

・自身のオブジェクトIDを表示

・Testclassのオブジェクト生成後、オブジェクトIDの表示、methodB methodCの実行を行っている。

※実行結果

```
21310332
methodA call
hello world !!
21310332
```

継承・カプセル化④

・スーパークラスの情報参照の実行例(superの使用)

```
class Eng
  def hello
    print("Hello !! ¥n")
  end
end

class Jap < Eng
  def hello
    super
    print("こんにちは！ ¥n")
  end
end

obj. = Jap.new
obj.hello
```

・サブクラスのオーバーライドされたメソッドの中で『super』を実行すると、スーパークラスの同名のメソッドが実行される。

※実行結果

```
Hello !!
こんにちは！
```


継承・カプセル化⑤

・アクセス修飾子

Rubyでは、『アクセス修飾子』を用いることで、クラス内に定義したメソッドにアクセス制限を設定する事が可能。

・アクセス修飾子の構文と解説

```
class HogeClass
  def methodA()
    :
  end
  private :methodA

  def methodB()
    :
  end
  protected :methodB
end
```

・メソッドのアクセス制限は、クラス中に

アクセス修飾子 メソッドと同名のシンボル

の形式で構文を記述する事で設定できる。

設定できるアクセス修飾子には用途に応じた以下の3種が用意されている。

- ・**public** ...アクセス制御無し。クラスをインスタンス化した際、オブジェクト名.メソッド名の形式(レシーバ形式)でメソッドの実行(アクセス)が可能。
アクセス修飾子を指定しない場合はアクセス制限はpublicになる。
- ・**private** ...レシーバ形式での実行(アクセス)が不可。(=クラス外からのメソッド実行が出来ない。クラス内の別のメソッドからのみ実行可。)
- ・**protected** ...レシーバ形式での実行(アクセス)が不可。但し、メソッドが所属するクラス内からならレシーバ形式での実行可。

継承・カプセル化⑥

・アクセス修飾子の使用例

```
class Testclass
  def primethod()
    puts "private method"
  end
  private :primethod

  def promethod()
    puts "protected method"
  end
  protected :promethod

  def pubmethod()
    primethod
    promethod
    self.promethod
    hoge = Testclass.new
    hoge.promethod
  end
end

obj = Testclass.new
obj.pubmethod
```

・privateメソッドに関数形式でアクセス(実行)。

・protectedメソッドに関数形式でアクセス。

・protectedメソッドにレシーバ形式でアクセス。

・protectedメソッドが所属するクラス内からレシーバ形式でアクセス。

・クラスの外部からpublicメソッドにレシーバ形式でアクセス。

※実行結果

```
private method
protected method
protected method
protected method
```

モジュール・Mix-in①

・モジュールとは

Rubyでは、任意の処理の実行に必要な要素(定数・メソッド等)をまとめ、1つの塊として定義することが可能である。この塊を『**モジュール**』と呼ぶ。

モジュールは、「処理を1つに纏める」という概念がクラスと酷似しているが、以下に列挙するようなクラスとの違いがある。

- ①クラスは「情報と機能」の塊であるのに対し、モジュールは「機能」のみの塊である。
- ②モジュールからはインスタンスを生成することが出来ない。(モジュールにインスタンスの概念は無い。)
- ③モジュールは継承できない

モジュール・Mix-in②

・モジュールの定義と使用方法

モジュールについて、定義の構文と使用方法を下記する。

・モジュール定義の構文とプログラムからの呼び出し

```

module Taxmodule
  TAXRATE = 0.05
  def calc( price )
    ret = price * TAXRATE
    ret.round
  end
  module_function :calc
end

price = 5000
tax = Taxmodule.calc(price)

puts "#{price}の消費税は#{tax}円です。”

```

※実行結果

```
5000円の消費税は250円です。
```

・モジュールの定義は

module モジュール名 ~ end

の形式で構文を記述する。尚、命名規則として、モジュール名の先頭は大文字にする事。

・モジュール内で定義したメソッド(ここではメソッドcalc)はそのままでは外部からの呼び出しが出来ない。メソッドをモジュール関数として外部に公開する際は、

module_function メソッドと同名のシンボル

の記述を行う必要がある。

・モジュール関数の実行は**モジュール名.メソッド名**の形式で行う。また、**モジュール名.定数名**の形式でモジュール内で定義した定数の参照が可能。

モジュール・Mix-in③

・Mix-inによる機能拡張

任意のクラスに指定したモジュールの機能を追加する事で、クラスの機能拡張が可能。この機能拡張手法を『Mix-in』と言う。

・Mix-inの実行例

```
module Testmod
  def hello
    puts "hello world !!"
  end
end

class Testcls
  include Testmod
  :
end
obj = Testcls.new
obj.hello
```

・クラス内にて、

include モジュール名

と記述する事で、指定したモジュールの内容(メソッドや定数)を、クラスの機能として取り込むことが可能。クラス継承とは異なり、1つのクラスに複数のモジュールをMix-inも可能。

・Testmodモジュールのモジュール関数helloをTestclsのメソッドとして呼び出す事が可能。なお、Mix-inを行う場合、モジュール関数helloをmodule_functionで外部公開する必要はない。

シンボル①

・シンボル

シンボルとは・・・ Symbolクラスのインスタンス。Symbolクラスは指定した文字列と1対1で対応するオブジェクトを生成する。

・シンボルの生成

```
x = :ruby
y = :ruby
z = :java
```

・『:』の後に任意の文字列を記述する事で、その文字列に対応したシンボルオブジェクトを生成できる。

このプログラムの場合、

『:ruby』, 『:java』という2つのシンボルオブジェクトが生成される。

シンボルオブジェクトは文字列と1対1に対応するので、プログラム中のxとyは同一のオブジェクトを示していることになる。

シンボル②

・シンボルの特徴

生成されたSymbolオブジェクトは指定した文字列と1対1で対応するユニークな値を持つ。

・ユニーク値の生成

```
puts :ruby.to_i
puts :ruby.to_i
puts :java.to_i
```

・生成されたシンボルのユニーク値を取得するには、シンボル名.to_iを実行する。

実行結果

```
10377
10377
10385
```

・各シンボルのユニーク値。(数値は実行環境によって異なる)
1, 2行目はどちらも文字列“ruby”のシンボル値を表示しているのと同じ値になる。
尚、この値はobject_id()メソッドで取得できるオブジェクトIDとは別物。

シンボル③

・シンボルの利用方法

シンボルは数値と同等の物として扱われるため、文字列を直接使用するより、メモリや実行速度の面で効率的になる。

・使用例: ハッシュのキーとして利用

```
hash1 = { :key1 => 3, :key2 => 4, :key3 => 5}

sum = hash1[:key1] + hash1[:key2] + hash1[:key3]

puts “valueの合計は#{sum}”
```

予約語①

・予約語一覧

● コメント系

BEGIN・・・複数行コメント開始

END・・・複数行コメント終了

● 条件分岐系

if・・・条件式

elsif・・・ifで真でなかった場合に
評価される条件式

else・・・if,elsifで真でなかった場合に
評価される条件式

unless・・・ifの条件を否定したもの

case・・・式の評価結果により処理を
分岐させる (Javaではswitch)

when・・・case節との評価結果を比較

then・・・条件式の直後に記述 (省略可)

● 擬似変数

true・・・真の値をとる

false・・・偽の値をとる

nil・・・偽の値をとる

self・・・現在のインスタンスを参照する

__FILE__・・・実行されているプログラムの
ファイル名

__LINE__・・・実行されているプログラムの
ファイル内行番号

● 例外処理系

begin・・・例外処理の対象コード開始 (Javaではtry)

end・・・例外処理の対象コード終了

rescue・・・例外が発生した時に実行される式を
記述 (Javaではcatch)

retry・・・rescue内で記述し、
begin節から処理を再実行する

ensure・・・式が終了する前に必ず実行される
式を記述 (Javaではfinally)

予約語②

● 繰り返し系

for・・・繰り返し処理を行う

while・・・繰り返し処理を行う

until・・・条件式が真になるまで繰り返しを行う

do・・・for, whileで使用 (省略可)

break・・・繰り返し処理を途中で抜ける

redo・・・繰り返し処理をやり直す

next・・・繰り返しの次の回の処理を実行する

in・・・繰り返しにおいて、

Enumerableオブジェクトを指定する

return・・・メソッドの実行を終了する

● 定義系

class・・・クラスを定義する

def・・・メソッドを定義する

alias・・・既に存在するメソッドに
別の名前を割り当てる

undef・・・メソッド定義を取り消す

module・・・モジュールを定義する

defined?・・・式が定義されていなければ、偽を返す

super・・・オーバーライドメソッドを呼び出す

yield・・・ブロック付きメソッド内で
ブロックを呼び出す

● 論理演算子

and・・・論理積

or・・・論理和

not・・・否定

正規表現①

・正規表現とは

文章の中からあるパターンにマッチする部分があるかどうかを調べる事は比較的多く使われる。完全に一致した部分があるかどうかを調べるだけではなく、より複雑なパターンを作成するために用意されたのが正規表現であり、正規表現で用意されている構文や特殊な文字を組み合わせる事で、複雑な条件を持つ検索パターンを簡潔に定義することが可能である。

正規表現②

・正規表現の書き方

```
/R.by/ =~ "Rubyセミナー"
```

文字列を/と/で囲むと、囲まれた部分が文字列のパターンを表す正規表現オブジェクトになる。

パターンマッチ演算子「 `=~` 」は、マッチングが成功したら、マッチングした部分の位置を返す。
先頭の文字の位置が0。
マッチングしなかった場合にはnilを返す。

```
s = Regexp.new("a.c")
```

正規表現オブジェクトを作るには、Regexp(レグエクスプ)クラスのnewメソッドを使用する。

実行例

```
s = Regexp.new("e.*o")
puts s =~ "hello"
```

実行結果

```
1
```

正規表現③

・正規表現の一例

```
[abc] a b cとマッチ
[a-z] abcdefg..zとマッチ
[a-zA-Z] abcdefg..z ABC..Zとマッチ
[^a] a以外の文字
(...) 正規表現のグルーピング Capture
(a|b) a or b
a* 直前の正規表現と0回以上マッチ
a? 0回(なし)か1回マッチ
a+ 1回以上マッチ
a{3} 3回繰り返し時にマッチ
a{3,} 3回以上繰り返し時にマッチ
a{3,6} 3回以上 6回以下マッチ
a*? 0回以上マッチ(最短一致)
a+? 1回以上マッチ(最短一致)
```

これはあくまで一例であり、

詳細は、正規表現の書籍などを参照すること。

32/51

正規表現④

・正規表現のオプション

正規表現の意味を少し変化させるためのオプションについて、

ここではオプション“x”と“i”を紹介する。

```
name = /shiori #名前/x
puts name =~ "shiori"
```

オプション“x”を使うと正規表現中で、「#」の後にコメントをつけることが可能。

実行結果

```
0
```

```
name = /SHIORI/i
puts name =~ "shiori"
```

オプション“i”を使うと大文字と小文字の区別がされなくなる。

実行結果

```
0
```

また、「/SHIORI #名前/xi」という様に複数のオプションを指定することも可能。

正規表現⑤

・正規表現のオプションの一覧

- s ...Shift_JISとしてマッチ
- e ...EUC-JPとしてマッチ
- u ...UTF-8としてマッチ
- n ...1バイト文字としてマッチ
- i ...大文字小文字を区別しない
- x ...パターン内の空白と、「#」より後ろを無視
- m ...複数行マッチ
(「.」が改行文字にもマッチするようになる)
- o ...式展開は一度のみ行う

ライブラリ・Timeクラス①

・requireメソッド

大抵のプログラミング言語では、別々のファイルに分割されたプログラムを組み合わせ、1つのプログラムとして利用するための機能を持っており、このように他のプログラムから読み込んで利用するためのプログラムをライブラリと呼ぶ。

Rubyのプログラムの中でライブラリを読み込むには、requireメソッドを使用する。

require “利用したいファイル名”

「”」の中にファイル名を書く。
また、ファイル名の「.rb」を省略することができる。

ライブラリ・Timeクラス②

例えば、ファイル名apple.rbのオブジェクトをrequireメソッドで利用する場合は以下のようなになる。

apple.rb

```
def apple
  puts "apple"
end
```

second_apple.rb

```
require "apple"
apple()
```

← ファイルapple.rbを読み込む。
← apple.rb内のappleメソッドを使用。

Rubyには、多くの便利なライブラリが標準で付属している。これらを利用する場合にもrequireメソッドを利用する。
どのようなライブラリがあるかは、リファレンスを参照。

ライブラリ・Timeクラス③

・Timeクラス

Timeクラスは時刻を扱うクラス。

現在の時刻を取得したり任意の時刻を設定したりする事が可能。

```
puts Time.new
#=> Mon Oct 12 22:08:17 +0900 2009
```

← Timeオブジェクトを作成するには「new」メソッド
作成されたTimeクラスのオブジェクトは作成した現在時刻が格納されている。

```
t = Time.now
#=> Mon Oct 12 22:08:18 +0900 2009
```

← 「now」でも同様。

ライブラリ・Timeクラス④

・時刻のフォーマット

時刻をフォーマットに従った文字列にしたいときには、`strftime()` メソッドを使う。

実行例

```
t = Time.now
p t.strftime("%Y/%m/%d %H:%M:%S")
```

実行結果

```
"2006/04/15 07:07:49"
```

実行例

```
t = Time.now
p t.strftime("%a %b %d %H:%M:%S %Z %Y")
```

実行結果

```
"Sat Apr 15 07:07:49 JST 2006"
```

実行例

```
t = Time.now
p t.to_s
```

実行結果

```
"Sat Apr 15 07:07:49 JST 2006"
```

ライブラリ・Timeクラス⑤

・フォーマット文字列

`strftime()` メソッドでは以下のようなフォーマット文字列が使用可能。

%A	曜日の名称
%a	曜日の省略名
%B	月の名称
%b	月の省略名
%c	日付と時刻
%d	日(01~31)
%H	24時間制の時(00~23)
%I	12時間制の時(00~12)
%j	年中の通算日(00~365)
%M	分(00~59)
%m	月を表す数字(01~12)
%p	午前または午後 (AM,PM)
%S	秒(00~60)

%U	週を表す数(00~53) 日曜日が週の始まり
%W	週を表す数(00~53) 月曜日が週の始まり
%w	曜日を表す数 日曜日が0(0~6)
%X	時刻
%x	日付
%Y	西暦を表す数
%y	西暦の下2桁(00~99)
%Z	タイムゾーン(JSTなど)
%z	タイムゾーン(+0900など)
%%	%をそのまま出力

ライブラリ・Timeクラス⑥

時刻の要素に関するメソッド

年・月・日など取得した時刻の要素を求めることも可能。

<code>t = Time.now</code>	
<code>p t.year</code>	← 年 => 2012
<code>p t.month</code>	← 月 => 3
<code>p t.day</code>	← 日 => 4
<code>p t.hour</code>	← 時 => 5
<code>p t.min</code>	← 分 => 12
<code>p t.sec</code>	← 秒 => 11
<code>p t.to_i</code>	← 1970年1月1日0時からの秒数 => (1147637531)
<code>p t.wday</code>	← 週の何日目か(日曜日を0とする) => 6
<code>p t.mday</code>	← 月の何日目か(dayメソッドと同じ) => 4
<code>p t.yday</code>	← 年の何日目か(1月1日を1とする) => 135
<code>p t.zone</code>	← タイムゾーン => "JST"

コマンドライン引数①

コマンドライン引数

コマンドラインでプログラムを実行するとき、引数を指定して実行することも可能。

(コマンドライン引数)

```
>ruby test.rb apple orange lemon
```

ソースファイル名

コマンドライン引数
引数を半角スペースで区切って並べる。

コマンドライン引数の値は、「ARGV」という特殊配列に格納される。

コマンドライン引数②

ARGVは、配列となっており、指定された引数のうち、1番目の引数が、ARGV[0]に、2番目がARGV[1]に、順次ARGV[2], ARGV[3]・・・と続く。

コマンドラインでの実行例

```
>ruby test.rb apple orange lemon
```

プログラム例

```
name1 = ARGV[0]
name2 = ARGV[1]
name3 = ARGV[2]
puts name1, name2, name3
```



表示結果

```
apple
orange
lemon
```

※ただし、コマンドライン引数から取得したデータは文字列になっているため、これを計算に使うときには数値に変換する必要がある。
文字列を整数にするには「to_i」メソッドを使用。

ファイル入出力・ディレクトリ操作①

・ファイルopen/close

Rubyのファイル入出力はIOクラスのサブクラスであるFileクラスを用いる。

ファイルを開く

```
sample_file = open( "sample_file.txt", "r" )
```

"r": モード
モードに関しては後述

ファイルを閉じる

```
sample_file.close
```

ファイル入出力・ディレクトリ操作②

・ファイルopen/close

モード

"r"	: 読み取りモード
"w"	: 書き込みモード(既存ファイル上書き)
"a"	: 追記モード
"r+"	: 読み書きモード(ファイル先頭から)
"w+"	: 読み書きモード(既存ファイル上書き)
"a+"	: 読み書きモード(ファイル末尾から)

ファイル入出力・ディレクトリ操作③

・ファイルopen/close

ブロックを使った方法

実行例

```
open("sample_file.txt"){|f|
  while line = f.gets
    処理
  end
}
```

ブロック終了時にファイルは自動で閉じられるためclose不要

ファイル入出力・ディレクトリ操作④

・入力操作

入かに用いられるメソッド例(1)

gets

ファイルの内容を1行ずつ読み込む。
ファイル末尾(EOF)を読み込むと nil を返す。

getc

データを1文字(1バイト)読み込む。
ファイル末尾(EOF)を読み込むと nil を返す。

ファイル入出力・ディレクトリ操作⑤

・入力操作

入かに用いられるメソッド例(2)

read(length)

データを length バイト読み込み、出力する。
ファイル末尾(EOF)を読み込むと nil を返す。

read

length を省略したときはファイル全体を読み込む。
ファイル末尾(EOF)を読み込むと "" を返す。

ファイル入出力・ディレクトリ操作⑥

・入力操作

入かに用いられるメソッド例(3)

readline

ファイルの内容を1行読み込む。
ファイル末尾(EOF)を読み込むとエラーを返す。

readlines

ファイルの内容を一度に読み込み、
各行を配列に格納する。
ファイル末尾(EOF)を読み込むと [] を返す。

ファイル入出力・ディレクトリ操作⑦

・出力操作

出かに用いられるメソッド例

puts

通常の puts と同等。

putc

通常の putc と同等。

print

通常の print と同等。

ファイル入出力・ディレクトリ操作⑧

・ファイル操作

メソッド例

```
File.rename(from, to)
```

ファイルの名前を変更する。

削除に失敗した場合はエラーを返す。

```
File.delete(filename)
```

```
File.unlink(filename)
```

ファイルを削除する。

削除に失敗した場合はエラーを返す。

ファイル入出力・ディレクトリ操作⑨

・ディレクトリ操作

Rubyのディレクトリ操作には Dir クラスを用いる。

メソッド例

```
Dir.pwd
```

カレントディレクトリを返す。

```
Dir.mkdir(path)
```

path で指定した新しいディレクトリを作成する。

```
Dir.rmdir(path)
```

path で指定したディレクトリを削除する。

問題1

Ruby における偽の値として正しいものすべてを選択してください。(二つ選択)

- 1. 0
- 2. false
- 3. ""
- 4. nil
- 5. NULL

回答	

問題2

以下のコードを実行したときの出力として、正しいものをひとつ選んでください。

```
t850 = -4.0
x = t850 <= -6.0 ? 'snow' : 'rain'
puts x
```

- 1. -4.0
- 2. -6.0
- 3. -10.0
- 4. snow
- 5. rain

回答

問題3

以下のような出力になるコードがあります。

に入る適切な記述をすべて選択してください。(二つ選択)

```
a = %w(a b c d e f)
p 
```

出力

```
["c", "d", "e"]
```

- 1. a[2, 3]
- 2. a[2, 4]
- 3. a[2...-1]
- 4. a[2..-1]
- 5. a[-4, 4]

回答	

問題4

以下のコードを実行した時の結果として正しいものをひとつ選択してください。

```
a = 10
b = 10
if !(a - b) then
  puts "hello"
else
  puts "good-bye"
end
```

- 1. hello
- 2. good-bye
- 3. nil
- 4. 何も表示されない

回答

問題5

以下のコードを実行したときの結果として正しいものをひとつ選択してください。

```
p "bar" * 2 ** 2
```

1. "barbarbarbar"
2. 実行時にエラーとなる
3. "bar4"
4. "bar*2**2"

回答
<input type="checkbox"/>

問題6

ハッシュから値だけを取り出すコードがあります。
□に入る適切な記述をひとつ選択してください。

```
h = {"month" => 2, "day" => 15, "year" => 1989}  
p h. □
```

出力

```
[2, 15, 1989]
```

1. only_values
2. values
3. values_only
4. value

回答
<input type="checkbox"/>

問題7

Rubyの変数名として誤っているものをひとつ選択してください。

1. @a
2. @@b
3. \$c
4. \$\$d

回答
<input type="checkbox"/>

問題8

以下のコードを実行したときの結果として、正しいものをひとつ選択してください。

```
puts "ABCDEF" [3...-1]
```

1. DE
2. DEF
3. BC
4. CD
5. CDEF

回答
<input type="checkbox"/>

問題9

以下のような出力になるコードがあります。
□に入る適切な記述をひとつ選択してください。

```
p ({"Red" => 0xff0000,
    "Green" => 0x00ff00,
    "Blue" => 0x0000ff
  }
). □
```

1. measure
2. dimensions
3. items
4. length
5. depth

出力

3

回答

問題10

以下のコードを実行したときの出力として正しいものをひとつ選択してください。

```
even_number = 101 % 2 == 0
if even_number == true
  p "偶数"
elsif
  p "奇数"
end
```

1. "偶数"
2. "奇数"
3. 何も表示されない
4. nil
5. 実行時にエラーになる

回答

問題11

以下のコードを実行したときの結果として正しいものをひとつ選択してください。

```
URL = "http://www.example.jp/"
URL = URL.sub!('http:', 'https:')
puts URL
```

1. http://www.example.jp/と表示される
2. URLは定数であるため警告が表示され、https://www.example.jp/と表示される
3. URLは定数であるため警告され、http://www.example.jp/と表示される
4. URLは定数であるため値は変わらず、http://www.example.jp/と表示される
5. 例外が発生し実行が中断する

回答

問題12

以下のコードを実行したときの結果として正しいものをひとつ選択してください。

```
begin
  x = 1 / 0 # ZeroDivisionError
rescue
  print "A"
rescue StandardError
  print "B"
rescue ZeroDivisionError
  print "C"
else
  print "D"
end
```

1. Aと表示される
2. Bと表示される
3. Cと表示される
4. Dと表示される
5. BCと表示される

回答

問題13

以下のコードを実行したときの出力として正しいものをひとつ選択してください。

```
x = 4..8
p x.to_a
```

1. [4, 5, 6, 7, 8]
2. 45678
3. 12
4. "4..8"
5. エラーが発生する

回答

問題14

以下の選択肢のコードの中から、実行時にエラーとなるものをすべて選択してください
(二つ選択)

1. puts "foo" + "bar"
2. puts "foo" - "bar"
3. puts "foo" * 3
4. puts "%x" / 255
5. puts "%x" % 255

回答	

問題15

以下のコードを実行したときの出力として正しいものをひとつ選択してください。

```
p "foo\nbar\nbaz".split(/\n/, 2)
```

1. ["foo", "bar", "baz"]
2. ["foo", "bar"]
3. ["foo", "bar\nbaz"]
4. ["bar", "baz"]
5. ["baz"]

回答

問題16

以下のコードを実行したときの出力として正しいものをひとつ選択してください。

```
def foo(x, *y)
  p x, y
end
foo(1, 2, 3, 4)
```

1. 1
"2, 3, 4"
2. [1]
[2, 3, 4]
3. 1
[2, 3, 4]
4. 1
(2, 3, 4)
5. 1
2, 3, 4

回答

問題17

文字列クラスのメソッド delete(str) は、文字列から str に含まれる文字を取り除きます。

以下のコードを実行したときの出力として正しいものをひとつ選択してください。

```
puts "0123456789-".delete("0-28-")
```

1. 34567-
2. 345679
3. 01234567890-
4. 1345679

回答

問題18

以下のような出力になるコードがあります。

に入る適切な記述をひとつ選択してください。

```
x = [1, 2, 3, 4, 5]
y = [3, 4, 5, 6]
p 
```

出力

[1, 2, 6]

1. $(x + y) - (x * y)$
2. $(x | y) - (x \& y)$
3. $(X - Y).abs$
4. $(y - x).abs$
5. $(x / y) * [6] * [6]$

回答
<input type="text"/>

問題19

ヒアドキュメントを使用して文字列を出力しようとしています。

に入る適切な記述をひとつ選択してください。

```
def foo
  lang = "Ruby"
  puts 
  We are #{lang} programmer.
  END
end

foo
```

出力

We are Ruby programmer.

1. <<END
2. <<"END"
3. <<'END'
4. <<-END
5. <<-'END'

回答
<input type="text"/>

問題20

以下のような出力になるコードがあります。

に入る適切な記述をすべて選択してください。(二つ選択)

```
ary = [1, 2, 3, 4]
p ary.  {|item| item ** 2}
```

出力

[1, 4, 9, 16]

1. slice
2. replace
3. map
4. flatten
5. collect

回答	
<input type="text"/>	<input type="text"/>

問題1

以下のコードを実行したときの出力として正しいものをひとつ選択してください。

```
p Time.now + 200.0
```

1. 実行時の日付・時刻から200日後の日付・時刻が表示される
2. 実行時の日付・時刻から200分後の日付・時刻が表示される
3. 実行時の日付・時刻から200秒後の日付・時刻が表示される
4. 実行時の日付・時刻から200ミリ秒後の日付・時刻が表示される
5. エラーとなる

回答
<input type="text"/>

問題2

以下のような出力になるコードがあります。
に入る適切な記述をひとつ選択してください。

```
t = Time.local(1988, 12, 23)  
puts 
```

出力
1988/12

1. t.strftime("%y/%m")
2. t.strftime("%Y/%m")
3. t.strftime("%y/%M")
4. t.strftime("%Y/%M")

回答
<input type="text"/>

問題3

コマンドラインから実行されたときのひとつめの引数を表示するプログラムfoo.rbがあります。
に入る適切な記述をひとつ選択してください。

```
puts 
```

コマンドの実行
ruby -w foo.rb bar

1. ARGV[0]
2. ARGV[1]
3. ARGV[2]
4. ARGV[3]
5. ARGV[4]

出力
bar

回答
<input type="text"/>

問題4

次の正規表現にマッチしない文字列をすべて選択してください。(二つ選択)
¥A は文字列の先頭を示すメタ文字、¥S は非空白文字を示すメタ文字です。

```
/¥A¥S[AUPQ][^AEFN]./
```

1. "AUPQ"
2. "AUXN"
3. "ASAS"
4. "AXFE"
5. "AUPA"

回答	
<input type="text"/>	<input type="text"/>

問題5

「Ruby」あるいは「ruby」のいずれかの文字列のみにマッチする正規表現をすべて選択してください。（二つ選択）

- 1. /¥A[Rr]uby¥z/
- 2. /¥ARuby|ruby¥z/
- 3. /¥AR|ruby¥z/
- 4. /¥A[Rr][u][b][y]¥z/

回答	

問題6

以下のコードを実行したときの結果として正しいものをひとつ選択してください。

```
p "int main(int argc, char **argv)".scan(/¥w+/.size
```

- 1. 1
- 2. 4
- 3. 5
- 4. 6
- 5. 7

回答

問題7

選択肢に記述されたコードを読み、例外が発生することなく変数 x の値が表示されるものをすべて選択してください（二つ選択）

- 1. x = 1
def foo
 puts x
end
foo
- 2. def foo
 if false then
 x = 10
 end
 puts x
end
foo
- 3. def foo
 x = \$x
 puts x
end
foo
- 4. def foo
 3.times {puts x}
 puts x
end
foo

回答	

問題8

以下のコードを実行した結果の出力として適切なものをひとつ選択してください。
Time クラスのメソッド mktime は local と同じ働きをします。

```
t = Time.mktime(1998, 9, 20, 00, 00)  
puts t.strftime("%Y/%M")
```

- 1. 1998/09
- 2. 1998/00
- 3. 98/09
- 4. 98/00

回答

問題9

二つの文字列が等しいかどうかを調べる文字列クラスのメソッド eql? と、二つのオブジェクトが同一のものであるかを調べるメソッド equal? があります。以下のコードを実行したときの出力として正しいものをひとつ選択してください。

```
a = "asdf"
b = "asdf"
puts a.eql?(b)
puts a.equal?(b)
```

- 1. true 2. true 3. false 4. false
- true false false true

回答

問題10

以下の演算子のうち、再定義可能なものをひとつ選択してください。

- 1. && 2. .. 3. and
- 4. = 5. +

回答

問題11

Fooクラス定義している以下の四つのコードのうち、全く同じ意味であるものをすべて選択してください。（二つ選択）

- 1. class Foo ; end
- 2. class Foo < Object ; end
- 3. class Foo < BasicObject ; end
- 4. class Foo < Kernel ; end

回答	

問題12

以下のコードを実行したときの結果として正しいものをひとつ選択してください。

```
class Integer
  def to_heisei
    self - 1988
  end
end

puts 2009.to_heisei
```

- 1. 実行時にエラーとなる
- 2. 21と表示される
- 3. -1998と表示される
- 4. 何も出力されない

回答

問題13

以下のRubyの演算子の中で再定義可能なメソッドとして定義されているものをすべて選択してください。(二つ選択)

1. [] 2. && 3. = 4. <=>

回答	

問題14

以下のコードを実行したときの結果として正しいものをひとつ選択してください。

```
class Object
  def foo
    self.upcase
  end
end

puts "abcd".foo
```

1. 実行時にエラーとなる
2. ABCDと出力される
3. nilと出力される
4. 何も出力されない

回答

問題15

以下のような出力になるコードがあります。

に入る適切な記述をひとつ選択してください。

```
  
p sin(PI / 2)
```

出力
1.0

1. require 'Math'
2. require Math
3. include 'Math'
4. include Math

回答

問題16

選択肢に記述されたコードを読み、例外が発生することなく変数 x の値が表示されるものをすべて選択してください。(二つ選択)

1.

```
x = 1
def foo
  puts x
end
foo
```
2.

```
def foo
  x = 10 if false
  puts x
end
foo
```
3.

```
def foo
  x=0
  (0..10).each do |i|
    puts x
  end
end
foo
```
4.

```
def foo
  for i in 1..10
    x += i
  end
  puts x
end
foo
```
5.

```
def foo
  (1..10).each do |i|
    x = 0
  end
  puts x
end
foo
```

回答	

問題17

ファイルfooの内容を読み込み、その内容をすべて大文字に変換したうえでファイルfooの末尾に追加するプログラムがあります。

に入る適切な記述をひとつ選択してください。

```
open("foo", ) do |f|
  f.write(f.read.upcase)
end
```

1. "a"
2. "r"
3. "r*"
4. "r+"
5. "r+w"

回答

問題18

以下のプログラムは、まずVectorクラスを定義しています。
それから配列に格納されたVectorクラスを、lengthメソッドの値で昇順にソートしたいと
思います。
この目的を達成するために、クラス定義の空欄に必要なコードをひとつ選択してください。

```
class Vector
  attr_accessor :x, :y
  def initialize(x = 0, y = 0)
    @x = x
    @y = y
  end
  def length
    Math::sqrt(@x ** 2 + @y ** 2)
  end
  def to_s
    "(#{@x}, #{@y})"
  end
  
end

arr = [ ]
arr << Vector.new(2, 2)
arr << Vector.new(3, 2)
arr << Vector.new
arr.sort.each do |item|
  puts item
end
```

出力

```
(0, 0)
(2, 2)
(3, 2)
```

1. def <=> other
length <=> other.length
end
2. include Sortable
3. include Enumerable
4. def sort(self, other)
self.length <=> other.length
end

回答

問題19

text.txtの内容を1行ずつ読み込んで表示し、その後終了するプログラムがあります。

に入る適切な記述をひとつ選択してください。

```
file = open("test.txt")
begin
  line_no = 1
  while true
    line = file. 
    printf "%5d: %s", line_no, line
    line_no += 1
  end
rescue EOFError
end
file.close
```

1. gets
2. readline
3. readlines
4. read

回答
<input type="text"/>

問題20

カレントディレクトリ内の、特定のワイルドカードに一致するファイルの一覧を取得して表示するコードがあります。

に入る適切な記述をすべて選択してください。（二つ選択）

```
files = 
files.each {|file| puts file }
```

1. Dir.glob("*.txt")
2. Dir.entries("*.txt")
3. Dir.open("*.txt")
4. Dir["*.txt"]
5. Dir.ls("*.txt")

回答	
<input type="checkbox"/>	<input type="checkbox"/>

問題1

Ruby における偽の値として正しいものすべてを選択してください。(二つ選択)

1. 0 2. false 3. ""
4. nil 5. NULL

Rubyでは、nil と false を除いてすべて 真 となります

回答	
2	4

問題2

以下のコードを実行したときの出力として、正しいものをひとつ選んでください。

```
t850 = -4.0  
x = t850 <= -6.0 ? 'snow' : 'rain'  
puts x
```

1. -4.0 2. -6.0 3. -10.0
4. snow 5. rain

3項演算子 $a ? b : c$ は if a then b else c と同等です。
条件式 $(-4.0 \leq -6.0)$ は 偽 ですので、rain が表示されます。

回答
5

問題3

以下のような出力になるコードがあります。

に入る適切な記述をすべて選択してください。(二つ選択)

```
a = %W(a b c d e f)  
p   
["c", "d", "e"]
```

1. a[2, 3]
2. a[2, 4]
3. a[2..-1]
4. a[2..-1]
5. a[-4, 4]

出力

["c", "d", "e"]

回答	
1	3

%w は文字列配列を生成する記法です。
スペースで区切られた字句を配列の連続する要素とします。
大文字 %W は式展開可、小文字 %w は式展開不可です。
a[2]から要素を3つ表示しますので 1 と、
a[2]からa[-2]までを表示しますので 3 が正解です。

問題4

以下のコードを実行した時の結果として正しいものをひとつ選択してください。

```
a = 10
b = 10
if !(a - b) then
  puts "hello"
else
  puts "good-bye"
end
```

1. hello
2. good-bye
3. nil
4. 何も表示されない

if 文の条件式 !(a-b) に関して
まず、(a-b)の計算を行いますので
条件式は !0 となります。
Ruby では 0 は 真 ですので
!0 は 偽 になります。

回答
2

問題5

以下のコードを実行したときの結果として正しいものをひとつ選択してください。

```
p "bar" * 2 ** 2
```

1. "barbarbarbar"
2. 実行時にエラーとなる
3. "bar4"
4. "bar*2**2"

演算子の優先順位の問題です。
* 乗算 と、** べき乗
では、** の方が優先度が高く
p "bar" * 4
となり、文字列 bar を 4回
表示します。

回答
1

問題6

ハッシュから値だけを取り出すコードがあります。
に入る適切な記述をひとつ選択してください。

```
h = {"month" => 2, "day" => 15, "year" => 1989}
p h. 
```

出力

```
[2, 15, 1989]
```

1. only_values
2. values
3. values_only
4. value

ハッシュから値だけを取り出すメソッド
→ values
キーだけを取り出すメソッド
→ keys

回答
2

問題7

Rubyの変数名として誤っているものをひとつ選択してください。

1. @a 2. @@b 3. \$c 4. \$\$d

@インスタンス変数、@@クラス変数、\$グローバル変数
\$\$は変数名として使用できません。

回答
4

問題8

以下のコードを実行したときの結果として、正しいものをひとつ選択してください。

```
puts "ABCDEF" [3...-1]
```

1. DE 2. DEF 3. BC
4. CD 5. CDEF

部分文字列を取り出します。
範囲演算子 ... は終点を含みません。
よって、1 が正解です。

回答
1

問題9

以下のような出力になるコードがあります。
□に入る適切な記述をひとつ選択してください。

```
p ({ "Red" => 0xff0000,  
  "Green" => 0x00ff00,  
  "Blue" => 0x0000ff  
  }  
). □
```

1. measure
2. dimensions
3. items
4. length
5. depth

length はハッシュの要素の数を
返すメソッドです

出力
3

回答
4

問題10

以下のコードを実行したときの出力として正しいものをひとつ選択してください。

```
even_number = 101 % 2 == 0  
if even_number == true  
  p "偶数"  
elsif  
  p "奇数"  
end
```

演算子の優先順位の問題です。
= 代入 と % 剰余 と == 等号
では、% → == → = の優先順位となり
even_number = 1 == 0
even_number = false
と解釈します。

1. "偶数"
2. "奇数"
3. 何も表示されない
4. nil
5. 実行時にエラーになる

回答
2

問題11

以下のコードを実行したときの結果として正しいものをひとつ選択してください。

```
URL = "http://www.example.jp/"
URL = URL.sub!('http:', 'https:')
puts URL
```

1. http://www.example.jp/と表示される
2. URLは定数であるため警告が表示され、https://www.example.jp/と表示される
3. URLは定数であるため警告され、http://www.example.jp/と表示される
4. URLは定数であるため値は変わらず、http://www.example.jp/と表示される
5. 例外が発生し実行が中断する

回答
2

sub!メソッドは、文字列中で引数1の内容にマッチした最初の部分を引数2の内容へ破壊的に置き換えます。
URLは大文字（定数）のため、変更すると警告メッセージが表示されます。

問題12

以下のコードを実行したときの結果として正しいものをひとつ選択してください。

```
begin
  x = 1 / 0 # ZeroDivisionError
rescue
  print "A"
rescue StandardError
  print "B"
rescue ZeroDivisionError
  print "C"
else
  print "D"
end
```

elseは例外が発生しなかったとき

1. Aと表示される
2. Bと表示される
3. Cと表示される
4. Dと表示される
5. BCと表示される

回答
1

begin - rescue - end による例外処理です。
begin内でエラーが発生したとき、rescueの処理を実行します。
rescueの後にエラータイプを指定することで、捕捉するエラーを限定することが可能ですが、指定が無い場合はすべてのエラーを捕捉します。
(StandardErrorのサブクラスである全ての例外)

問題13

以下のコードを実行したときの出力として正しいものをひとつ選択してください。

```
x = 4..8  
p x.to_a
```

1. [4, 5, 6, 7, 8]
2. 45678
3. 12
4. "4..8"
5. エラーが発生する

`to_a` メソッドはオブジェクトを配列に変換した結果を返します。

回答
1

問題14

以下の選択肢のコードの中から、実行時にエラーとなるものをすべて選択してください
(二つ選択)

1. puts "foo" + "bar" 文字列"foo"と"bar"を連結します
2. puts "foo" - "bar" エラー
3. puts "foo" * 3 文字列"foo"を3回出力します
4. puts "%x" / 255 エラー
5. puts "%x" % 255 255を16進数表記します。

回答
2 4

問題15

以下のコードを実行したときの出力として正しいものをひとつ選択してください。

```
p "foo¥nbar¥nbaz".split(/¥n/, 2)
```

1. ["foo", "bar", "baz"]
2. ["foo", "bar"]
3. ["foo", "bar¥nbaz"]
4. ["bar", "baz"]
5. ["baz"]

`split`メソッドは引数1をセパレータとして引数2個に分割します
ここでは、`¥n`をセパレータとして2つに分割しています。

回答
3

問題16

以下のコードを実行したときの出力として正しいものをひとつ選択してください。

```
def foo(x, *y)
  p x, y
end
foo(1, 2, 3, 4)
```

1. 1
"2, 3, 4"
2. [1]
[2, 3, 4]
3. 1
[2, 3, 4]
4. 1
(2, 3, 4)
5. 1
2, 3, 4

メソッドに渡したい引数が一定でない場合は
可変長引数を利用します。
第1引数は実引数、第2引数を配列にして渡します。

回答
3

問題17

文字列クラスのメソッド delete(str) は、文字列から str に含まれる文字を取り除きます。

以下のコードを実行したときの出力として正しいものをひとつ選択してください。

```
puts "0123456789-".delete("0-28-")
```

1. 34567-
2. 345679
3. 01234567890-
4. 1345679

deleteメソッドで削除されるのは[0-2][8][-]となる。
よって"345679"

回答
2

問題18

以下のような出力になるコードがあります。

に入る適切な記述をひとつ選択してください。

```
x = [1, 2, 3, 4, 5]
y = [3, 4, 5, 6]
p 
```

出力

[1, 2, 6]

1. (x + y) - (x * y)
2. (x | y) - (x & y)
3. (X - Y).abs
4. (y - x).abs
5. (x / y) * [6] * [6]

- は差集合で、左辺の要素から右辺の要素を取り除いた配列を返します
& は積演算で、二つの配列に共通して含まれる要素を持つ配列を返します

回答
2

問題19

ヒアドキュメントを使用して文字列を出力しようとしています。

に入る適切な記述をひとつ選択してください。

```
def foo
  lang = "Ruby"
  puts 
  We are #{lang} programmer.
  END
end

foo
```

出力

We are Ruby programmer.

1. <<END
2. <<"END"
3. <<' END'
4. <<-END
5. <<-' END'

回答
4

ヒアドキュメントは指定された終了記号の直前までの複数行を文字列とするリテラルです。

終了記号は原則として行の先頭に記述します。

<<- で始めた場合に限り、空白文字を入れることができます。

問題20

以下のような出力になるコードがあります。

に入る適切な記述をすべて選択してください。（二つ選択）

```
ary = [1, 2, 3, 4]
p ary.  {|item| item ** 2}
```

出力

[1, 4, 9, 16]

1. slice
2. replace
3. map
4. flatten
5. collect

- 1: エラー。sliceはArray#[]と同じ
- 2: エラー。replaceは配列の内容を置換
- 3: [1, 4, 9, 16] collectと同じ
- 4: [1, 2, 3, 4]配列の配列を平滑化
- 5: [1, 4, 9, 16]mapと同じ

回答	
3	5

問題1

以下のコードを実行したときの出力として正しいものをひとつ選択してください。

```
p Time.now + 200.0
```

1. 実行時の日付・時刻から200日後の日付・時刻が表示される
2. 実行時の日付・時刻から200分後の日付・時刻が表示される
3. 実行時の日付・時刻から200秒後の日付・時刻が表示される
4. 実行時の日付・時刻から200ミリ秒後の日付・時刻が表示される
5. エラーとなる

回答
3

Timeクラスは単位を秒で扱います。

問題2

以下のような出力になるコードがあります。

に入る適切な記述をひとつ選択してください。

```
t = Time.local(1988, 12, 23)
puts 
```

出力

1988/12

1. t.strftime("%y/%m")
2. t.strftime("%Y/%m")
3. t.strftime("%y/%M")
4. t.strftime("%Y/%M")

Timeクラスのメソッド strftime
時刻をフォーマットして表示します。

%Y: 西暦を表す数

%y: 西暦の下2桁 (00-99)

%M: 分 (00-59)

%m: 月を表す数字 (01-12)

回答
2

問題3

コマンドラインから実行されたときのひとつめの引数を表示するプログラムfoo.rbがあります。

に入る適切な記述をひとつ選択してください。

```
puts 
```

コマンドの実行

```
ruby -w foo.rb bar
```

出力

bar

1. ARGV[0]

2. ARGV[1]

3. ARGV[2]

4. ARGV[3]

5. ARGV[4]

コマンドライン引数は
ARGV[0]から始まります。

回答
1

問題4

次の正規表現にマッチしない文字列をすべて選択してください。(二つ選択)

¥A は文字列の先頭を示すメタ文字、¥S は非空白文字を示すメタ文字です。

```
/¥A¥S[AUPQ][^AEFN]./
```

1. "AUPQ"
2. "AUXN"
3. "ASAS"
4. "AXFE"
5. "AUPA"

文字列の先頭文字が空白ではない → すべて

文字列の2文字目が A, U, P, Q のいずれか → 1, 2, 5

文字列の3文字目が A, E, F, N のいずれでもない → 1, 2, 5

回答	
3	4

問題5

「Ruby」あるいは「ruby」のいずれかの文字列のみにマッチする正規表現をすべて選択してください。(二つ選択)

- 1. /¥A[Rr]uby¥z/
- 2. /¥ARuby|ruby¥z/
- 3. /¥AR|ruby¥z/
- 4. /¥A[Rr][u][b][y]¥z/

¥A, ¥zはそれぞれ、文字列の先頭と末尾を示すメタ文字です。
2はRubyで始まる文字列、またはrubyで終わる文字列にマッチ
3は“R”で始まる文字列、またはrubyで終わる文字列にマッチ

回答	
1	4

問題6

以下のコードを実行したときの結果として正しいものをひとつ選択してください。

```
p "int main(int argc, char **argv)".scan(/¥w+/).size
```

- 1. 1
- 2. 4
- 3. 5
- 4. 6
- 5. 7

文字列“int main(int argc, char **argv)”を
1文字以上の英数字でマッチし、マッチした文字列を使って
配列を作成します。[int, main, int, argc, char, argv]

回答
4

問題7

選択肢に記述されたコードを読み、例外が発生することなく変数 x の値が表示されるものをすべて選択してください。(二つ選択)

- 1. x = 1
def foo
 puts x
end
foo
- 2. def foo
 if false then
 x = 10
 end
 puts x
end
foo
- 3. def foo
 x = \$x
 puts x
end
foo
- 4. def foo
 3.times {puts x}
 puts x
end
foo

- 1:ローカル変数 x はメソッド foo で未定義なのでエラー
- 2:ローカル変数 x は代入されていないので nil
- 3:グローバル変数 \$x がスクリプトのどこにも定義されていないので nil
- 4:ローカル変数 x が未定義なのでエラー

回答	
2	3

問題8

以下のコードを実行した結果の出力として適切なものをひとつ選択してください。
Time クラスのメソッド mktime は local と同じ働きをします。

```
t = Time.mktime(1998, 9, 20, 00, 00)
puts t.strftime("%Y/%M")
```

- 1. 1998/09
- 2. 1998/00
- 3. 98/09
- 4. 98/00

%Y は4桁西暦 %H は24時間
%m は2桁月 %I は12時間
%d は2桁日 %M は2桁分

回答
2

問題9

二つの文字列が等しいかどうかを調べる文字列クラスのメソッド eql? と、二つのオブジェクトが同一のものであるかを調べるメソッド equal? があります。以下のコードを実行したときの出力として正しいものをひとつ選択してください。

```
a = "asdf"
b = "asdf"
puts a.eql?(b)
puts a.equal?(b)
```

- 1. true 2. true 3. false 4. false
- true false false true

回答
2

a と b は文字列としては同じではあるが
オブジェクトは別のものとして定義される

問題10

以下の演算子のうち、再定義可能なものをひとつ選択してください。

- 1. && 2. .. 3. and
- 4. = 5. +

回答
5

オーバーライドできる演算子を選択します。
メソッドとして実装されている演算子が該当します。
3. +(4) #=> 7

問題11

Fooクラス定義している以下の四つのコードのうち、全く同じ意味であるものをすべて選択してください。（二つ選択）

- 1. class Foo ; end
- 2. class Foo < Object ; end
- 3. class Foo < BasicObject ; end
- 4. class Foo < Kernel ; end

回答	
1	2

クラス定義時にスーパークラスを省略すると
Objectクラスがスーパークラスになります。

問題12

以下のコードを実行したときの結果として正しいものをひとつ選択してください。

```
class Integer
  def to_heisei
    self - 1988
  end
end

puts 2009.to_heisei
```

- 1. 実行時にエラーとなる
- 2. 21と表示される
- 3. -1998と表示される
- 4. 何も出力されない

回答
2

既存のクラスにメソッドを追加したり、再定義したりできます。
ここではIntegerクラスにメソッド to_heiseiを追加しています。
ちょっと強引な方法です

問題13

以下のRubyの演算子の中で再定義可能なメソッドとして定義されているものをすべて選択してください。（二つ選択）

1. [] 2. && 3. = 4. <=>

回答	
1	4

問題14

以下のコードを実行したときの結果として正しいものをひとつ選択してください。

```
class Object
  def foo
    self.upcase
  end
end

puts "abcd".foo
```

1. 実行時にエラーとなる
2. ABCDと出力される
3. nilと出力される
4. 何も出力されない

Objectクラスでfooメソッドを定義したあとに文字列でfooメソッドを呼び出しています。StringはObjectを継承しているため、Objectで定義したfooを呼出可能です。

回答
2

問題15

以下のような出力になるコードがあります。
に入る適切な記述をひとつ選択してください。

```
  
p sin(PI / 2)
```

出力
1.0

1. require 'Math' 2. require Math
3. include 'Math' 4. include Math

Mathモジュールは三角関数や対数など、数学演算を行うメソッドが定義されているモジュールです。

回答
4

問題16

選択肢に記述されたコードを読み、例外が発生することなく変数 x の値が表示されるものをすべて選択してください。(二つ選択)

```
1. x = 1
   def foo
     puts x
   end
   foo
```

```
2. def foo
     x = 10 if false
     puts x
   end
   foo
```

```
3. def foo
     x=0
     (0..10).each do |i|
       puts x
     end
   end
   foo
```

```
4. def foo
     for i in 1..10
       x += i
     end
     puts x
   end
   foo
```

```
5. def foo
     (1..10).each do |i|
       x = 0
     end
     puts x
   end
   foo
```

- 1: 0-加変数 x が未宣言
- 2: nil を表示
- 3: 0 を 10 回表示
- 4: x が未定義なので + のリット`利用不可
- 5: 0-加変数 x が未宣言

回答	
2	3

問題17

ファイルfooの内容を読み込み、その内容をすべて大文字に変換したうえでファイルfooの末尾に追加するプログラムがあります。

に入る適切な記述をひとつ選択してください。

```
open("foo", ) do |f|
  f.write(f.read.upcase)
end
```

書き込みと読み込みの両方を行うには読み書き両用モードで開く必要があります。a+, r+, w+ のいずれかが該当しますが、w+ だとOPEN時にファイルを空にするため、末尾に追加することは出来ません。

- 1. "a"
- 2. "r"
- 3. "r*"
- 4. "r+"
- 5. "r+w"

回答
4

問題18

以下のプログラムは、まずVectorクラスを定義しています。
それから配列に格納されたVectorクラスを、lengthメソッドの値で昇順にソートしたいと
思います。
この目的を達成するために、クラス定義の空欄に必要なコードをひとつ選択してください。

```
class Vector
  attr_accessor :x, :y
  def initialize(x = 0, y = 0)
    @x = x
    @y = y
  end
  def length
    Math::sqrt(@x ** 2 + @y ** 2)
  end
  def to_s
    "(#{@x}, #{@y})"
  end
  空欄
end

arr = [ ]
arr << Vector.new(2, 2)
arr << Vector.new(3, 2)
arr << Vector.new
arr.sort.each do |item|
  puts item
end
```

出力

```
(0, 0)
(2, 2)
(3, 2)
```

1. def <=> other
length <=> other.length
end
2. include Sortable
3. include Enumerable
4. def sort(self, other)
self.length <=> other.length
end

Array#sort

配列の内容をソートします。
要素同士の比較には <=> 演算子を使用します。
sort はソートされた配列を生成して返します。

回答

1

問題19

text.txtの内容を1行ずつ読み込んで表示し、その後終了するプログラムがあります。
□に入る適切な記述をひとつ選択してください。

```
file = open("test.txt")
begin
  line_no = 1
  while true
    line = file. □
    printf "%5d: %s", line_no, line
    line_no += 1
  end
rescue EOFError
end
file.close
```

1. gets
2. readline
3. readlines
4. read

回答
2

- ◆ gets
→1行ずつ読み込み、ファイルの終端に達するとnilを返す
- ◆ readline
→1行ずつ読み込み、ファイルの終端に達するとErrorを返す
- ◆ readlines
→1行ずつ読み込み、各行を配列の要素として読み込む。
- ◆ read
→指定したバイト分だけ読み込み、ファイル終端に達すると空文字列を返す。
バイト数を指定しないとすべてのデータを返す。

問題20

カレントディレクトリ内の、特定のワイルドカードに一致するファイルの一覧を取得して表示するコードがあります。

に入る適切な記述をすべて選択してください。（二つ選択）

```
files =   
files.each {|file| puts file }
```

1. Dir.glob("*.txt")
2. Dir.entries("*.txt")
3. Dir.open("*.txt")
4. Dir["*.txt"]
5. Dir.ls("*.txt")

回答	
1	4

glob(pattern, flags=0) -> [Pathname]

[PARAM] pattern: ワイルドカードパターンです

[PARAM] flag: パターンマッチ時のふるまいを変化させるフラグを指定します

ワイルドカードの展開を行なった結果を、Pathname オブジェクトの配列として返します。

entries(path) -> [String]

[PARAM] path: ディレクトリのパスを文字列で指定します。

ディレクトリ path に含まれるファイルエントリ名の配列を返します。

[](*pattern) -> [String]

ワイルドカードの展開を行った結果を文字列の配列として返します。

パターンにマッチするファイルがない場合は空の配列を返します。

e-learning 事前準備手順

ここでは今回のe-learning講座を進行する上での事前準備を行います。
以下に1.から5.までの手順を示しているの、順に進行をお願いします。

1.Slackでワークスペースへの参加

下記は今回のe-learning用のSlackのURLです。e-learning講座における情報共有の為に活用します。
リンク先から参加をお願いします。参加にはメールアドレスを使用します。
Slackのアカウント登録をされていない方もリンク先から登録が可能です。

https://join.slack.com/t/oic-e-learning2020/shared_invite/zt-g9f4796o-hgrKSp6hPqImo0j9CYmj3Q

2.GitHubのアカウント作成

e-learningのコンテンツ内で使用するアカウントを作成します。
アカウントを既に取得されている方は、そちらを使っていただいても構いません。

※プランは\$0のフリープランを選択してください

作成方法は別紙「GitHubアカウント作成」をご参考ください

3.AWSアカウント登録

e-learningを進行するうえで必須となるアカウントです。
AWSのアカウント登録には、以下のものが必須となりますのでご準備した上で進めてください。

- ・クレジットカードまたはデビットカード
- ・SMSまたは携帯電話

使用方法によっては利用料金が発生する可能性があります。

登録前に必ずex.補足情報をご覧ください。

下記のリンクにAWSのアカウント取得手順が記載されていますので、そちらをご参考に進めてください。

AWSアカウント取得手順説明：<https://aws.amazon.com/jp/register-flow/>

4.Cloud9環境構築

e-learningを進行する為に使用する開発環境です。
設定内容によっては料金が発生するものがありますので、必ず手順通りに行ってください。
手順は別紙「Cloud9環境構築」をご参考ください

5.Ruby on Rails 環境構築

3のCloud9環境構築の終了直後からの続きになります。
Ruby on Railsの環境構築を行います。
手順は別紙「Ruby on Rails 環境構築」をご参考ください

ex.補足情報

例えば、新規にご契約いただいた方は12ヶ月無料ですが、仮想環境の利用時間は750時間/月です。これは、ひと月目一杯使っても24時間 x 31日=744時間となり、750時間/月には到達しませんが、仮想環境を2つ同時に動かすことがあれば、それぞれ動作させた時間が加算されますので制限時間を超過して利用料金が発生する可能性があります。

12ヶ月を超過した場合、1つの仮想環境だけを利用していても、その環境で利用しているディスクスペースや環境を動作させている時間に応じて従量課金されます。

Ruby on Railsのサーバーを動作させたまま(つまりrails serverを実行したまま)だと、インスタンスが休止状態にならない可能性も考えられますので、学習時間以外は、コマンドを入力するターミナル画面で、動作させているrails serverに対して、[CTRL]キーと[C]キーを同時に押して、Ruby on Railsのサーバーを停止されることをおすすめします。

例)

```
$ rails s(<--このコマンドでrails serverを起動したと思います)
```

```
=> Booting Puma
```

```
=> Rails 5.1.7 application starting in development
```

```
=> Run 'rails server -h' for more startup options
```

```
Puma starting in single mode...
```

```
* Version 3.12.1 (ruby 2.6.3-p62), codename: Llamas in Pajamas
```

```
* Environment: development
```

```
* Listening on tcp://localhost:8080
```

```
Use Ctrl-C to stop(<-- [CTRL]キーと[C]キーを同時に押すと停止できますよ)
```

のように表示されていると思いますので、[CTRL]キーと[C]キーを同時に押して停止します。

また、Cloud9の環境を構成する際、AWSのおすすめ構成で構築いただいたと思いますが、インスタンスの種類で、t2.micro(1GiB RAM : 1vCPU)はFree-tierですが、それ以外の構成で構築し直すと無料期間でも利用料金が発生しますので、環境をがおかしくなって、削除、再作成などをされる際は、お気をつけください。

GitHubアカウント作成

20/7/20時点



1.以下のURLへアクセスします

<https://github.co.jp/>

2.画像と同じ画面が表示されるので、画面中央の赤枠で囲っている緑の「GitHubに登録する」ボタンを押下します



3.ユーザー名・Eメール・パスワードを入力し、機械認証をクリアした後、「create account」をクリックします。



Join GitHub

Create your account

Username *

Email address *

Password *

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)


Email preferences

Send me occasional product updates, announcements, and offers.

Verify your account

このパズルを解いて、実在の人物であることを証明してください。

[検証開始](#)



[Create account](#)

By creating an account, you agree to the [Terms of Service](#). For more information about GitHub's privacy practices, see the [GitHub Privacy Statement](#). We'll occasionally send you account-related emails.

4.職業・プログラミングの経験・GitHubの使用目的の3つの質問項目があるのでそれぞれ回答してセットアップを完了してください。

この時、自動的に無料枠のFreePlanが選択されています。(画面左上)

Selected plan: Free

Welcome to GitHub

Wooohoo! You've joined millions of developers who are doing their best work on GitHub. Tell us what you're interested in. We'll help you get there.

- 1** What kind of work do you do, mainly?
 - Software Engineer (I write code)
 - Student (I go to school)
 - Product Manager (I write specs)
 - LUX & Design (I draw interfaces)
 - Data & Analytics (I write queries)
 - Marketing & Sales (I look at charts)
 - Teacher (I educate people)
 - Other (I do my own thing)
- 2** How much programming experience do you have?
 - None (I don't program at all)
 - A little (I'm new to programming)
 - A moderate amount (I'm somewhat experienced)
 - A lot (I'm very experienced)
- 3** What do you plan to use GitHub for?
(Select up to 3)
 - Learn to code
 - Learn Git and GitHub
 - Host a project (repository)
 - Create a website with GitHub Pages
 - Collaborating with my team
 - Find and contribute to open source
 - School work and student projects
 - Use the GitHub API
 - Other

I am interested in:
languages, frameworks, industries
We'll connect you with communities and projects that fit your interests.
For example: [rails](#) [wordpress](#) [contentful](#)

Complete setup

↓セットアップ完了後の画面



Please verify your email address

Before you can contribute on GitHub, we need you to verify your email address.

An email containing verification instructions was sent to **登録したメールアドレス**

[Resend verification email](#)

[Change your email settings](#)

5.登録したメールアドレス宛に確認メールが届くので、メール内の「verify email address」ボタンを押下すればアカウント登録が完了です。

Cloud9環境構築

cloud9はAWSで提供されているオンラインIDE(統合開発環境)です。

1.下記のリンクからCloud9へログインしてください。ログインにはAWSのアカウントを使用します。

<https://console.aws.amazon.com/cloud9>

ログイン時はルートユーザーを選択してください。



サインイン

ルートユーザー
無制限アクセスを必要とするタスクを実行するアカウント所有者。 [詳細はこちら](#)

IAM ユーザー
日常的なタスクを実行するアカウント内のユーザー。 [詳細はこちら](#)

ルートユーザーのEメールアドレス

username@example.com

次へ

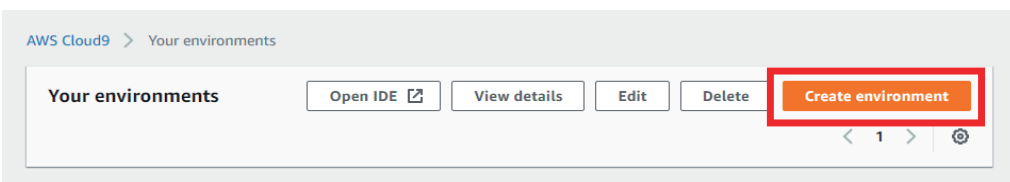
— AWSのご利用は初めてですか? —



**AWS のアカウントには 12 か月の
AWS 無料利用枠が含まれます**

Amazon EC2、Amazon S3、
Amazon RDS の使用が含まれます

提供規約の詳細については、aws.amazon.com/free をご覧ください。



3. 「Name」に開発環境の名前を入力します

下の「Description」は開発環境環境の説明文なので入力自由です。

AWS Cloud9 > Environments > Create environment

Step 1
Name environment

Step 2
Configure settings

Step 3
Review

Name environment

Environment name and description

Name
The name needs to be unique per user. You can update it at any time in your environment settings.

Limit: 60 characters

Description - Optional
This will appear on your environment's card in your dashboard. You can update it at any time in your environment settings.

Write a short description for your environment

Limit: 200 characters

Cancel **Next step**

4.開発環境の設定を行います。

- Environment type: Info: Create a new EC2 instance for environment (direct access)
Instance type: t2.micro (1 GiB RAM + 1 vCPU)
Platform: Amazon Linux
Cost-saving setting: After 30 minutes(default)

上の4つが選択されていることを確認して、右下の「Next step」ボタンをクリックします。

AWS Cloud9 > Environments > Create environment

Step 1
Name environment

Step 2
Configure settings

Step 3
Review

Configure settings

Environment settings

Environment type [Info](#)
Run your environment in a new EC2 instance or an existing server. With EC2 instances, you can connect directly through Secure Shell (SSH) or connect via AWS Systems Manager (without opening inbound ports).

- Create a new EC2 instance for environment (direct access)
Launch a new instance in this region that your environment can access directly via SSH.
- Create and run in remote in remote server (SSH connection)
Configure the secure connection to the remote server for your environment.

Instance type

- t2.micro (1 GiB RAM + 1 vCPU)
Free-tier eligible. Ideal for educational users and exploration.
- t3.small (2 GiB RAM + 2 vCPU)
Recommended for small-sized web projects.
- m5.large (8 GiB RAM + 2 vCPU)
Recommended for production and general-purpose development.
- Other instance type
Select an instance type.

t3.nano

Platform

- Amazon Linux
- Ubuntu Server 18.04 LTS

Cost-saving setting
Choose a predetermined amount of time to auto-hibernate your environment and prevent unnecessary charges. We recommend a hibernation settings of half an hour of no activity to maximize savings.

After 30 minutes (default)

IAM role
AWS Cloud9 creates a service-linked role for you. This allows AWS Cloud9 to call other AWS services on your behalf. You can delete the role from the AWS IAM console once you no longer have any AWS Cloud9 environments. [Learn more](#)

AWSServiceRoleForAWSCloud9

► Network settings (advanced)

No tags associated with the resource.

Add new tag

You can add 50 more tags.

Cancel Previous step Next step

5.ここまで手順通りに進んだら、右下の「Create environment」ボタンをクリック

AWS Cloud9 > Environments > Create environment

Step 1
Name environment

Step 2
Configure settings

Step 3
Review

Review

Environment name and settings

Name
eee

Description
No description provided

Environment type
EC2

Instance type
t2.micro

Subnet

Platform
Amazon Linux

Cost-saving settings
After 30 minutes (default)

IAM role
AWSServiceRoleForAWSCloud9 (generated)

We recommend the following best practices for using your AWS Cloud9 environment

- Use **source control and backup** your environment frequently. AWS Cloud9 does not perform automatic backups.
- Perform regular **updates of software** on your environment. AWS Cloud9 does not perform automatic updates on your behalf.
- **Turn on AWS CloudTrail in your AWS account** to track activity in your environment. [Learn more](#)
- Only share your environment with **trusted users**. Sharing your environment may put your AWS access credentials at risk. [Learn more](#)

Cancel Previous step Create environment

cloud9の環境構築はここまでになります。

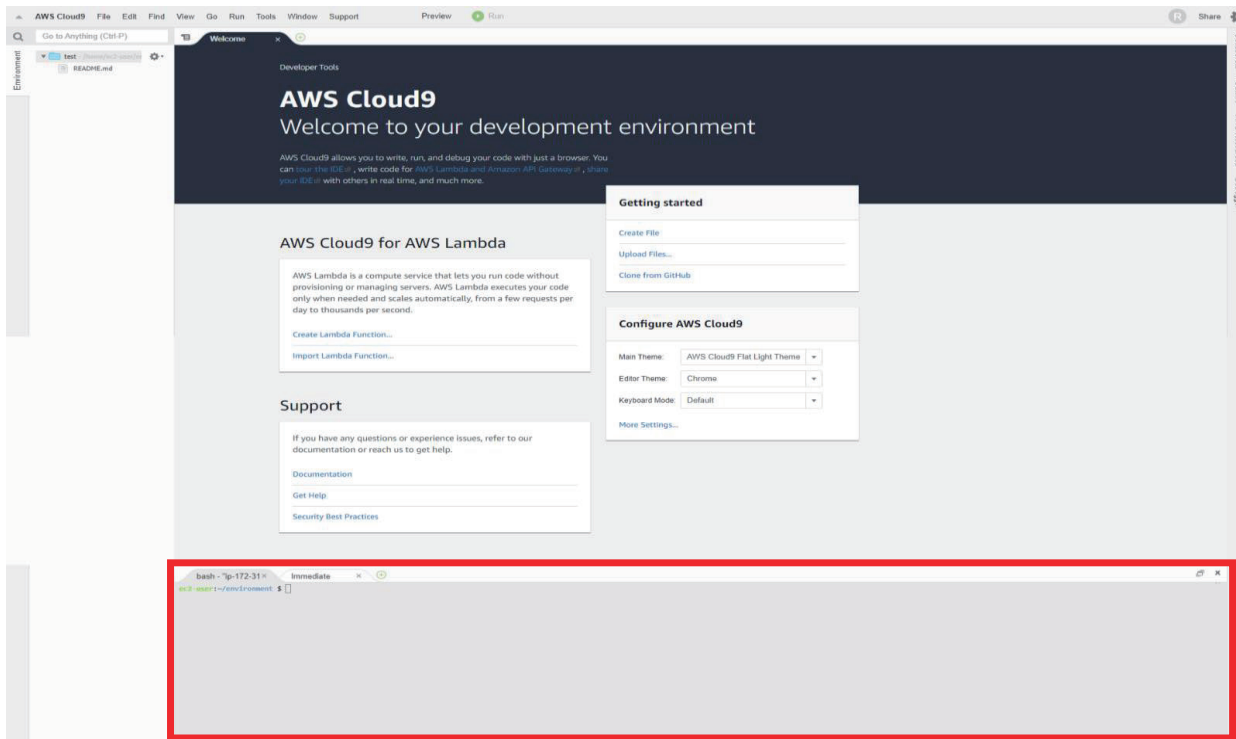
このあとも続きますので、Cloud9の画面はそのままにしてください

Ruby on Rails 環境構築

3のCloud9環境構築の終了直後からの続きになります。

1.前回の続きですと、以下の画面が表示されているかと思います。

この先では、赤枠で囲っている場所をターミナルと呼びます。



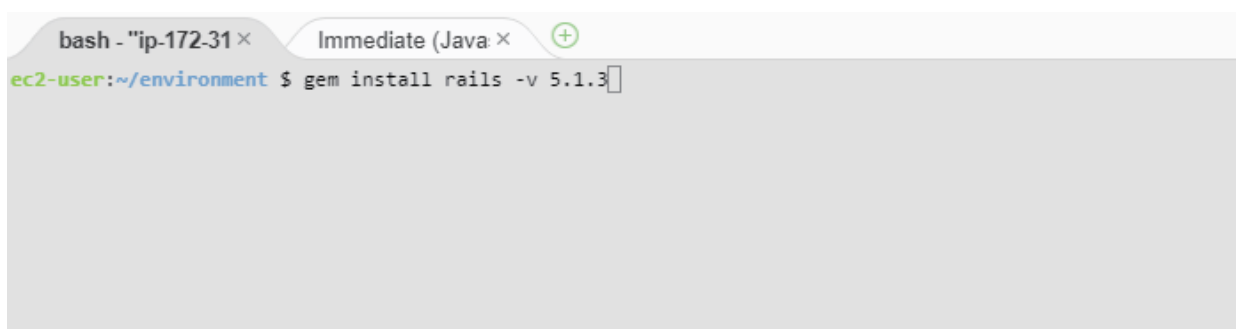
万が一ページを閉じたりしてしまった場合は、下記のURLから該当する環境を選択して「open IDE」をクリックしてください。

<https://console.aws.amazon.com/cloud9>

2.Ruby on Railsのバージョンを指定してインストールを行います。

画面下部のターミナルに、「ec2-user:~/environment \$」と表示されているので、\$の後ろに続いて下記のコマンドを入力してください(数分程度かかります)

コマンド: `gem install rails -v 5.1.3`



3.下記のコマンドを入力して、Ruby on Railsのプロジェクトを新規作成します。(数分程度かかります)

コマンド: rails new test01

```
bash - "ip-172-31" × Immediate (Java × +)
Parsing documentation for activejob-5.1.3
Installing ri documentation for activejob-5.1.3
Parsing documentation for actionmailer-5.1.3
Installing ri documentation for actionmailer-5.1.3
Parsing documentation for arel-8.0.0
Installing ri documentation for arel-8.0.0
Parsing documentation for activemodel-5.1.3
Installing ri documentation for activemodel-5.1.3
Parsing documentation for activerecord-5.1.3
RDoc detects ERB file. Skips it for compatibility:
lib/rails/generators/active_record/migration/templates/create_table_migration.rb
RDoc detects ERB file. Skips it for compatibility:
lib/rails/generators/active_record/migration/templates/migration.rb
Installing ri documentation for activerecord-5.1.3
Parsing documentation for rails-5.1.3
Installing ri documentation for rails-5.1.3
Done installing documentation for activesupport, erubi, actionview, actionpack, railties, nio4r, actioncable, activejob, action
mailer, arel, activemodel, activerecord, rails after 11 seconds
13 gems installed
ec2-user:~/environment $
ec2-user:~/environment $
ec2-user:~/environment $
ec2-user:~/environment $ rails new test01
```

4.プロジェクトの階層に移動する為、下記のコマンドを入力します。

コマンド: cd test01

```
bash - "ip-172-31" × Immediate (Java × +)
If you are starting a NEW Rails application, you can ignore this notice.

For more info see:
https://github.com/svenfuchs/i18n/releases/tag/v1.1.0

Post-install message from sass:

Ruby Sass has reached end-of-life and should no longer be used.

* If you use Sass as a command-line tool, we recommend using Dart Sass, the new
  primary implementation: https://sass-lang.com/install

* If you use Sass as a plug-in for a Ruby web framework, we recommend using the
  sassc gem: https://github.com/sass/sassc-ruby#readme

* For more details, please refer to the Sass blog:
  https://sass-lang.com/blog/posts/7828841

   run bundle exec spring binstub --all
* bin/rake: Spring inserted
* bin/rails: Spring inserted
ec2-user:~/environment $ cd test01
```

5. Railsのサーバーを起動します。下記のコマンドを入力してください。

コマンド: rails server

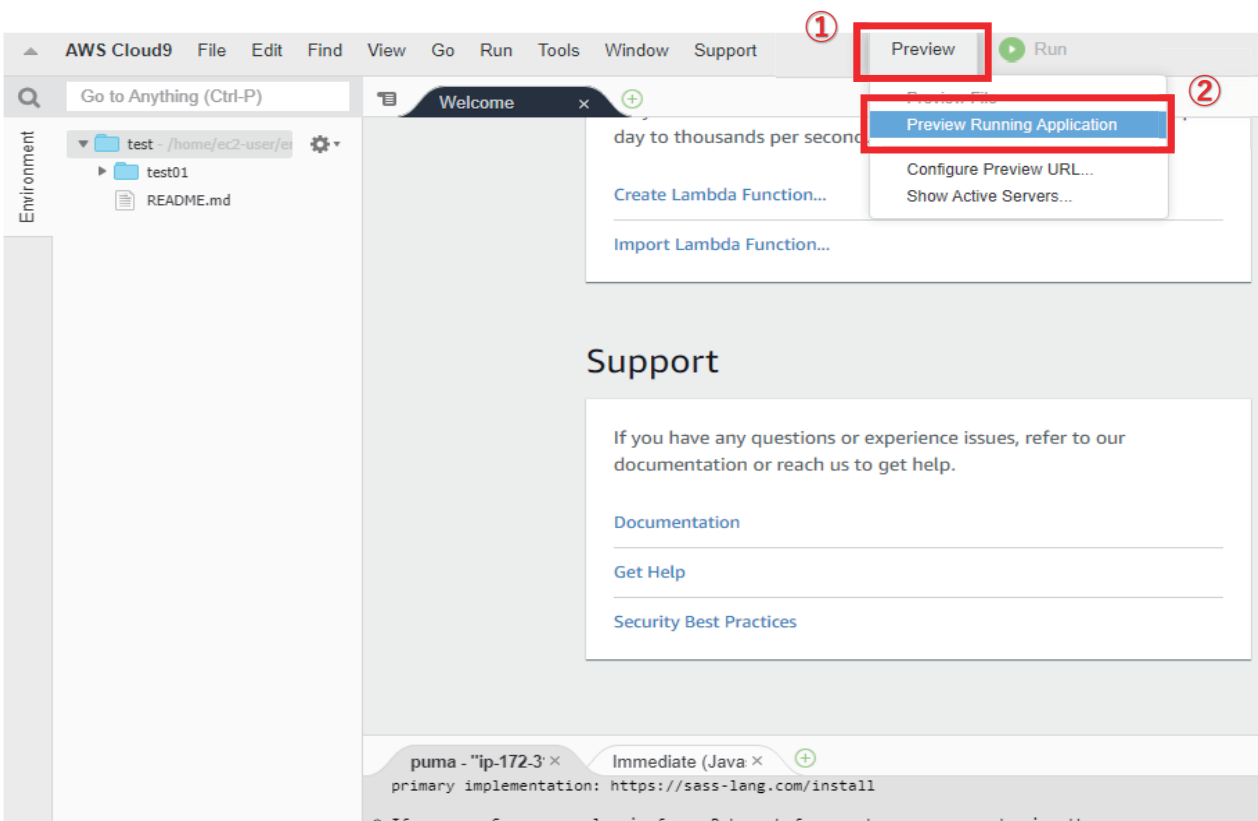
```
puma - "ip-172-3" × Immediate (Java × +)
primary implementation: https://sass-lang.com/install

* If you use Sass as a plug-in for a Ruby web framework, we recommend using the
  sassc gem: https://github.com/sass/sassc-ruby#readme

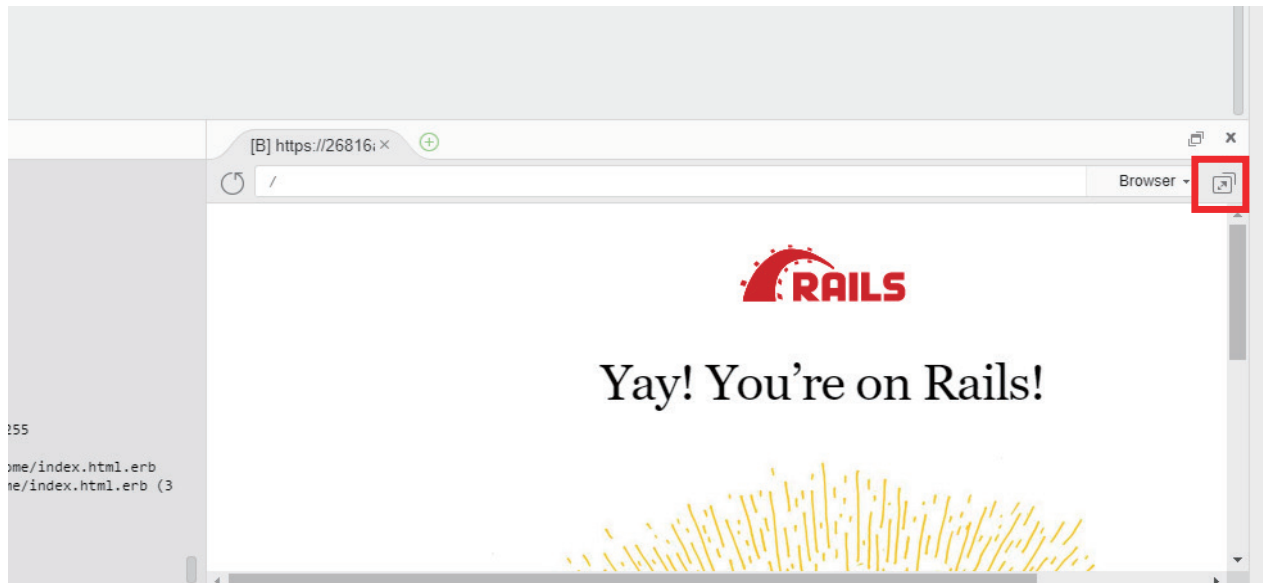
* For more details, please refer to the Sass blog:
  https://sass-lang.com/blog/posts/7828841

  run bundle exec spring binstub --all
* bin/rake: Spring inserted
* bin/rails: Spring inserted
ec2-user:~/environment $ cd test01
ec2-user:~/environment/test01 (master) $ rails server
=> Booting Puma
=> Rails 5.1.7 application starting in development
=> Run `rails server -h` for more startup options
Puma starting in single mode...
* Version 3.12.6 (ruby 2.6.3-p62), codename: Llamas in Pajamas
* Min threads: 5, max threads: 5
* Environment: development
* Listening on tcp://localhost:8080
Use Ctrl-C to stop
□
```

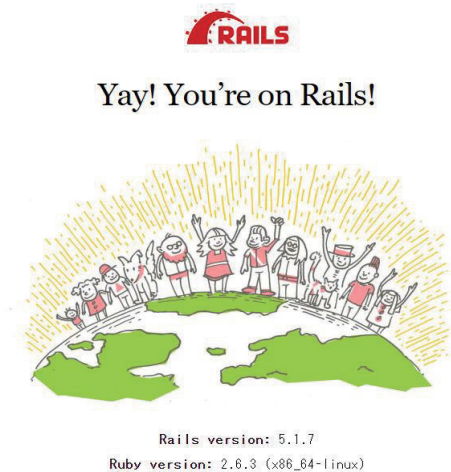
6.画面上部の「Preview」をクリックし、「Preview Running Application」をクリック



7.画面下にウィンドウが生成されるので、画像の赤枠部分の「四角いアイコン」をクリック



8.デフォルトで設定しているブラウザが開きますので、
画像の「Yay!You're on Rails!」が表示されれば環境構築完了です。



2020/08/21 (金)

Ruby on Rails 講義90分：内容

- scaffoldによる簡易Webアプリ制作
- 有名なRailsチュートリアル(<https://railstutorial.jp/>)をなぞりながらハンズオン進行
- 第4版 (Rails5.1) https://railstutorial.jp/chapters/toy_app?version=5.1#cha-a_toy_app
- チュートリアルのサンプルプログラムのバージョンと、Cloud9に環境構築したバージョンの違いがあるため、読み替え部分に注意

Cloud9

1. Railsアプリケーションの作成

```
$ cd ~/environment
$ rails new toy_app
$ cd toy_app/
```

2. Toyアプリケーション用のGemfileを編集

```
source 'https://rubygems.org'

gem 'rails', '5.1.7'           # チュートリアルから変更してます
gem 'puma', '3.9.1'
gem 'sass-rails', '5.0.6'
gem 'uglifier', '3.2.0'
gem 'coffee-rails', '4.2.2'
gem 'jquery-rails', '4.3.1'
gem 'turbolinks', '5.0.1'
gem 'jbuilder', '2.7.0'
gem 'sqlite3', '1.3.13'      # チュートリアルから変更してます

group :development, :test do
  # gem 'sqlite3', '1.3.13'   # チュートリアルから変更してます
  gem 'byebug', '9.0.6', platforms: :mri
end

group :development do
  gem 'web-console', '3.5.1'
  gem 'listen', '3.1.5'
  gem 'spring', '2.0.2'
  gem 'spring-watcher-listen', '2.0.1'
end
```

```
#group :production do          # チュートリアルから変更してます
#  gem 'pg', '0.20.0'
#end

# Windows環境ではtzinfo-dataというgemを含める必要があります
gem 'tzinfo-data', platforms: [:mingw, :mswin, :x64_mingw, :jruby]
```

3. gemをインストール (`--path` オプションでインストールパスを指定すると無難)

```
$ bundle install --path vendor/bundle
```

実行時に下記エラーが表示された場合は先に `bundle update spring` を実行してから再実行

```
You have requested:
  spring = 2.0.2

The bundle currently has spring locked at 2.1.0.
Try running `bundle update spring`
```

4. scaffoldによるUsersリソースの作成

```
$ rails generate scaffold User name:string email:string
```

ユーザーのモデル設計 (チュートリアル2.1.1)

users	
id	integer
name	string
email	string

5. scaffoldによるMicropostリソースの作成

```
$ rails generate scaffold Micropost content:text user_id:integer
```

マイクロポストのモデル設計 (チュートリアル2.1.2)

microposts	
id	integer
content	text
user_id	integer

6. データベースのマイグレーションを実行する

```
$ rails db:migrate
```

7. Applicationコントローラに `hello` アクションを追加

```
app/controllers/application_controller.rb
```

```
class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception

  def hello
    render html: "hello, world!"
  end
end
```

8. ルーティングを設定

```
config/routes.rb
```

```
Rails.application.routes.draw do
  resources :microposts
  resources :users
  root 'application#hello' # TOPページ
end
```

9. Webアプリケーション実行

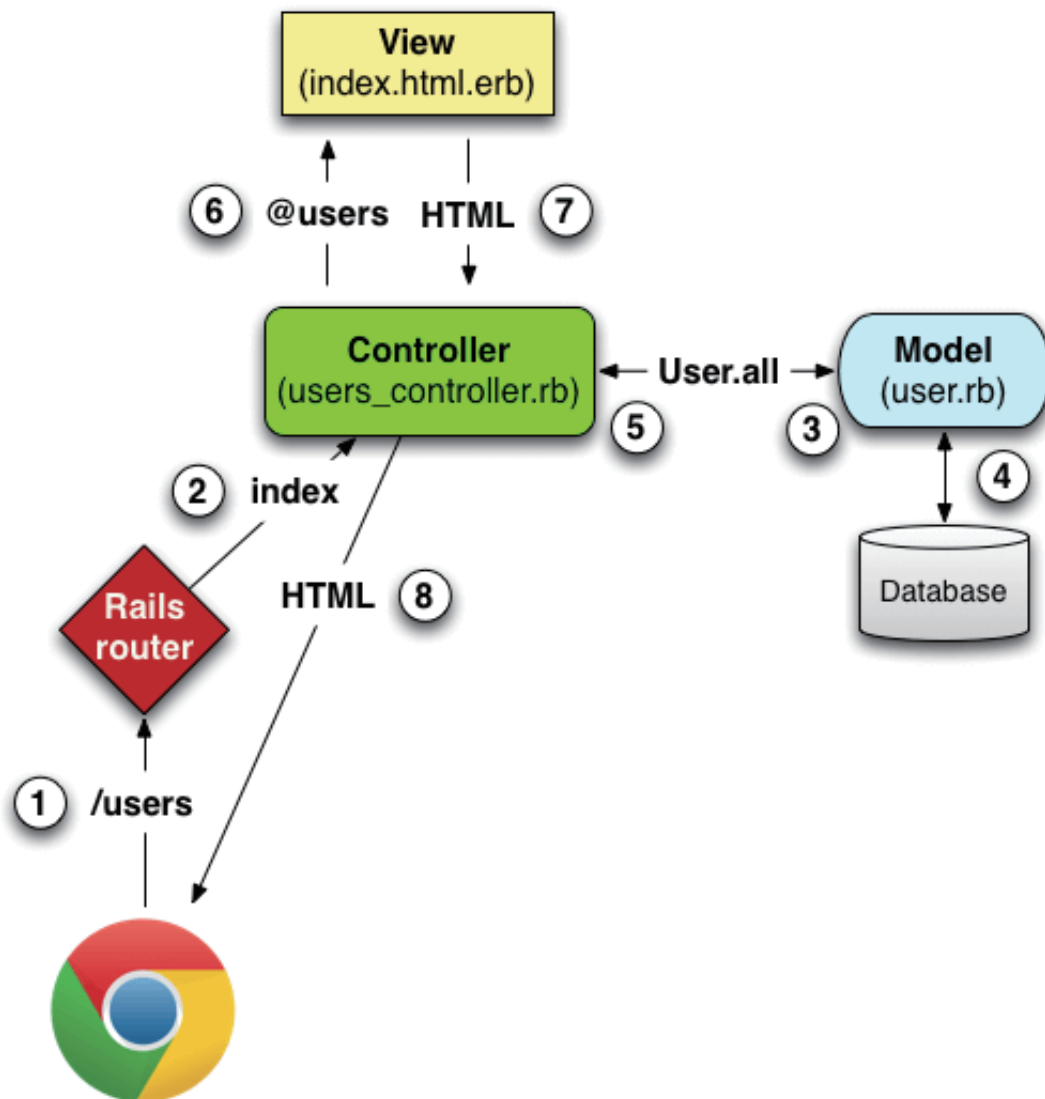
```
$ rails s
```

ブラウザから画面オープンする際のURL

- /users
- /microposts

10. MVC (Model-View-Controller)

「/users にあるindexページをブラウザで開く」 (チュートリアル2.2.2)



1. ブラウザから「/users」というURLのリクエストをRailsサーバーに送信する。
2. 「/users」リクエストは、Railsのルーティング機構 (ルーター) によってUsersコントローラ内の `index` アクションに割り当てられる。
3. `index` アクションが実行され、そこからUserモデルに、「すべてのユーザーを取り出せ」 (`User.all`) と問い合わせる。
4. Userモデルは問い合わせを受け、すべてのユーザーをデータベースから取り出す。
5. データベースから取り出したユーザーの一覧をUserモデルからコントローラに返す。
6. Usersコントローラは、ユーザーの一覧を `@users` 変数 (@はRubyのインスタンス変数を表す) に保存し、`index` ビューに渡す。
7. `index` ビューが起動し、ERB (Embedded RuBy: ビューのHTMLに埋め込まれているRubyコード) を実行してHTMLを生成 (レンダリング) する。
8. コントローラは、ビューで生成されたHTMLを受け取り、ブラウザに返す。

11. コントローラ

`index, show, new, edit, create, update, destroy` の7つのアクションメソッドが存在

`app/controllers/users_controller.rb`

```

class UsersController < ApplicationController
  before_action :set_user, only: [:show, :edit, :update, :destroy]

  # GET /users
  # GET /users.json
  def index
    @users = User.all
  end

  # GET /users/1
  # GET /users/1.json
  def show
  end

  # GET /users/new
  def new
    @user = User.new
  end

  # GET /users/1/edit
  def edit
  end

  # POST /users
  # POST /users.json
  def create
    @user = User.new(user_params)

    respond_to do |format|
      if @user.save
        format.html { redirect_to @user, notice: 'User was successfully
created.' }
        format.json { render :show, status: :created, location: @user }
      else
        format.html { render :new }
        format.json { render json: @user.errors, status:
:unprocessable_entity }
      end
    end
  end

  # PATCH/PUT /users/1
  # PATCH/PUT /users/1.json
  def update
    respond_to do |format|
      if @user.update(user_params)
        format.html { redirect_to @user, notice: 'User was successfully
updated.' }
        format.json { render :show, status: :ok, location: @user }
      end
    end
  end
end

```

```

    else
      format.html { render :edit }
      format.json { render json: @user.errors, status:
:unprocessable_entity }
    end
  end
end

# DELETE /users/1
# DELETE /users/1.json
def destroy
  @user.destroy
  respond_to do |format|
    format.html { redirect_to users_url, notice: 'User was successfully
destroyed.' }
    format.json { head :no_content }
  end
end

private
# Use callbacks to share common setup or constraints between actions.
def set_user
  @user = User.find(params[:id])
end

# Never trust parameters from the scary internet, only allow the white
list through.
def user_params
  params.require(:user).permit(:name, :email)
end
end

```

`index, show, new, edit` はビューのファイル名と対応する (`app/views` 内のファイルを確認)

12. ビュー

コントローラで宣言したインスタンス変数 (`@user`) はビューでも使用できる
HTMLに変換してブラウザに表示する

`app/views/index.html.erb` : indexアクションに対応しているビュー

```

<p id="notice"><%= notice %></p>

<h1>Users</h1>

<table>
  <thead>

```

```

<tr>
  <th>Name</th>
  <th>Email</th>
  <th colspan="3"></th>
</tr>
</thead>

<tbody>
<% @users.each do |user| %>
  <tr>
    <td><%= user.name %></td>
    <td><%= user.email %></td>
    <td><%= link_to 'Show', user %></td>
    <td><%= link_to 'Edit', edit_user_path(user) %></td>
    <td><%= link_to 'Destroy', user, method: :delete, data: { confirm:
'Are you sure?' } %></td>
  </tr>
<% end %>
</tbody>
</table>

<br>

<%= link_to 'New User', new_user_path %>

```

13. REST (チュートリアル コラム2.2)

HTTPリクエスト	URL	アクション	用途
GET	/users	index	すべてのユーザーを一覧するページ
GET	/users/1	show	id=1のユーザーを表示するページ
GET	/users/new	new	新規ユーザーを作成するページ
POST	/users	create	ユーザーを作成するアクション
GET	/users/1/edit	edit	id=1のユーザーを編集するページ
PATCH	/users/1	update	id=1のユーザーを更新するアクション
DELETE	/users/1	destroy	id=1のユーザーを削除するアクション

14. モデルを編集

`app/models/user.rb`

```
class Micropost < ApplicationRecord
  belongs_to :user # 1人のユーザーに複数のマイクロポストがある
  validates :content, length: { maximum: 140 }, presence: true
  # 最大文字数は140文字で、入力を必須とする
end
```

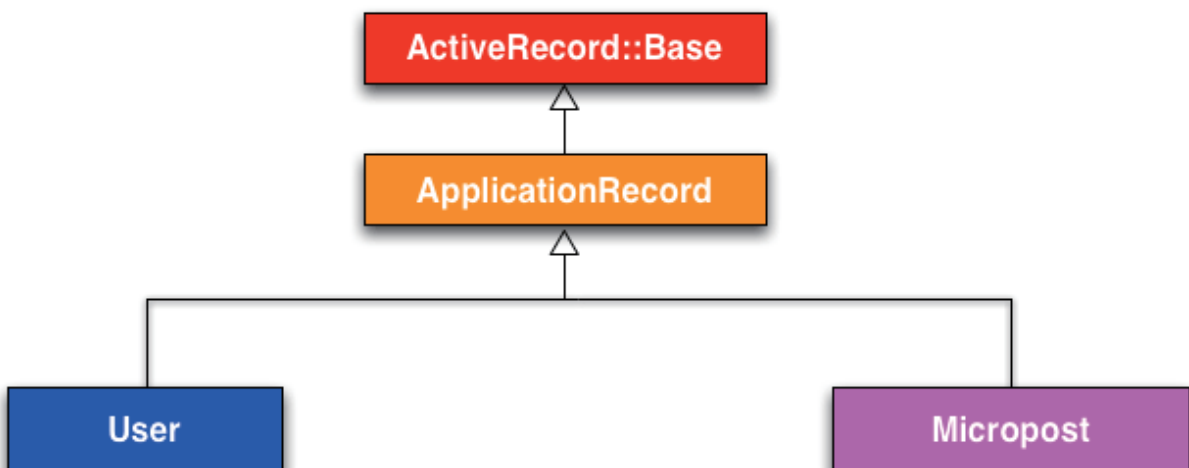
`app/models/micropost.rb`

```
class User < ApplicationRecord
  has_many :microposts # 1つのマイクロポストは1人のユーザーにのみ属する
end
```

データベースのリレーション (チュートリアル2.3.3)

microposts			users		
id	content	user_id	id	name	email
1	First post!	1	1	Michael Hartl	mhartl@example.com
2	Second post	1	2	Foo Bar	foo@bar.com
3	Another post	2			

クラスの継承 (チュートリアル2.3.4)



問題1

a~eに当てはまる語句を選択肢から選んで文章を完成させてください

(a.)は、ソフトウェア工学において迅速かつ適応的にソフトウェア開発を行う(b.)開発手法群の総称である。プロセスやツールよりも(c.)を、包括的なドキュメントよりも(d.)を、契約交渉よりも顧客との協調を計画に従うことよりも(e.)を 価値とする。

- | | |
|---------------|------------------|
| 1: 変化への対応 | 6: 動くソフトウェア |
| 2: オブジェクト指向開発 | 7: ウォーターフォール開発 |
| 3: 個人と対話 | 8: アジャイルソフトウェア開発 |
| 4: 軽量な | |
| 5: 重量な | |

回答 a: b: c: d: e:

問題2

次の1~5のうちアジャイル開発の特徴ではないものを選択してください。

- 1: フェーズに進んでしまうと後戻りができない
- 2: 高い顧客満足
- 3: リスクを最小限にできる
- 4: 迅速かつ継続的なリリース
- 5: 変更要求に柔軟に対応する

回答 ()

問題3

テスト駆動開発(TDD)について説明した文章です。

a~dに当てはまる語句を選択肢1~7の中から選んで文章を完成させてください。

(a.)よりも先に(b.)を作成する。(a.)は、そのテストをパスするように(c.)に作成する。テストはテストコードとしてプログラミングし(d.)することで、変更コストを抑制することができる。

- | | |
|--------|---------|
| 1: 自動化 | 6: シンプル |
| 2: 複雑 | 7: テスト |
| 3: 永続化 | |
| 4: 網羅的 | |
| 5: 実装 | |

回答 a: b: c: d:

問題4

a~dに当てはまる語句を選択肢1~5の中から選んで文章を完成させてください。

KPTは、Keep Problem Try の頭文字からそう呼ばれています。

KEEP: (a.)
(b.): うまくいかなかったこと、問題点
(c.): (d.)

- 1: KEEP
- 2: 改善策、次にためしたいこと
- 3: TRY
- 4: よかったこと、続けたいこと
- 5: PROBLEM

回答 a: b: c: d:

問題5

ソフトウェア開発において計画を狂わせる要因になりえるものを選択肢1~6の中から選んでください。(複数選択)

1. プロジェクトリーダー予算管理の権限を持っていない
2. より良い実装手段が見つかった
3. 顧客がプロジェクトに参加しない
4. 契約時に当該ITプロジェクトにおける顧客の役割を規定していなかった
5. 開発開始後に新たな課題が見つかる
6. 途中で要件や仕様が変わる

回答 ()

問題6

1~4の中からスプリントバックログについて正しく説明したものを選んでください。

1. プロダクトバックログのうち 今回のスプリントで開発するストーリーを取り出し
そのストーリーを完成させるために必要なすべての作業項目を洗い出したもの
2. スプリントで実装できたバックログのポイントの合計
3. ポイントや残作業時間、残タスク数を使ってバーンダウンチャートを描くことでプロジェクトの進捗を見える化を行う
4. プロダクトに必要な機能・要件・要望・修正事項をリスト化したもの

回答 ()

問題7

a~dに当てはまる語句を選択肢1~4の中から選んで文章を完成させてください。

アナログツール vs デジタルツール

- ・全員が同じ場所で作業しているなら
自由度の高い(a.)ツールがオススメ
- ・色々マトリクスを取りたくなったら
(b.)に移行
- ・情報冷蔵庫にならないように注意が必要
- ・(c.)に(d.)を合わせるのではなく、
(d.)にマッチする(c.)を探す

1. アナログ 3. デジタル
2. 運用 4. ツール

回答 a: b: c: d:

問題8

継続的インテグレーションに適したツールを1~4の中から選択してください。(複数選択)

1. Redmine
2. Jenkins
3. CircleCI
4. Trello

回答 ()

令和2年度「専修学校リカレント教育総合推進プロジェクト」

技術者学び直し講座のモデルとなる IT エンジニアを対象とした e ラーニング講座開設およびガイドラインの実証

■実施委員会

- | | |
|---------|---|
| ◎ 原辺 隆吉 | 大阪情報コンピュータ専門学校 校長 |
| 村岡 好久 | 名古屋工学院専門学校 講師／一般社団法人 TukurouneMono 振興協会代表理事 |
| 谷口 英司 | 日本電子専門学校 情報ビジネスライセンス科科长 |
| 北原 聡 | 麻生情報ビジネス専門学校 校長代行 |
| 小幡 忠信 | 一般社団法人 Ruby ビジネス推進協議会 理事長 |
| 岡山 保美 | 株式会社ユニバーサル・サポート・システムズ 取締役 |
| 高畑 道子 | 一般社団法人女性と地域活性推進機構 理事 |
| 飯塚 正成 | 一般社団法人全国専門学校情報教育協会 専務理事 |

■事業実施分科会

- | | |
|---------|--|
| ◎ 岡山 保美 | 株式会社ユニバーサル・サポート・システムズ 取締役 |
| 呉本 能基 | 大阪情報コンピュータ専門学校 副校長 |
| 櫻井 健一 | 大阪情報コンピュータ専門学校 総合情報学部 |
| 清水 素彦 | 大阪情報コンピュータ専門学校 総合情報学部 |
| 菅野 崇行 | 吉田学園情報ビジネス専門学校 情報系学科主任 |
| 村岡 好久 | 名古屋工学院専門学校 講師／一般社団法人 TukurouneMono 振興協会代表理事 |
| 谷口 英司 | 日本電子専門学校 情報ビジネスライセンス科科长 |
| 北原 聡 | 麻生情報ビジネス専門学校 校長代行 |
| 大磯 洋明 | コーデソリューション株式会社 代表取締役 |
| 大園 博美 | 有限会社Aries 代表 |
| 川端 光義 | 株式会社アジャイルウェア 代表取締役
／一般社団法人 Ruby ビジネス推進協議会 理事 |
| 石丸 博士 | リバティ・フィッシュ株式会社代表取締役社長
／一般社団法人 Ruby ビジネス推進協議会 理事 |
| 高畑 道子 | 一般社団法人女性と地域活性推進機構 理事 |
| 吉岡 正勝 | 一般社団法人全国専門学校情報教育協会 |

■評価委員会

- | | |
|---------|----------------------------|
| ◎ 中野 秀男 | NHL 中野秀男研究所 代表／大阪市立大学 名誉教授 |
| 高畑 道子 | 一般社団法人女性と地域活性推進機構 理事 |
| 飯塚 正成 | 一般社団法人全国専門学校情報教育協会 専務理事 |

令和2年度「専修学校リカレント教育総合推進プロジェクト」

技術者学び直し講座のモデルとなる IT エンジニアを対象とした e ラーニング講座開設およびガイドラインの実証

スクーリング指導書

令和3年2月

学校法人大阪経理経済学園 大阪情報コンピュータ専門学校
〒543-0001 大阪府大阪市天王寺区上本町 6-8-4
TEL 06-6772-2233 FAX 06-6772-1272

●本書の内容を無断で転記、掲載することは禁じます。